ÉCOLE NATIONALE DES CHARTES

Paul, Hector Kervegan

Modélisation, enrichissement sémantique et diffusion d'un corpus textuel semi-structuré : le cas des catalogues de vente de manuscrits

Mémoire pour le diplôme de master

Technologies numériques appliquées à l'histoire

2022

Résumé

Introduction

J'ai choisi de structurer mon mémoire autour de plusieurs questions connexes, qui, à différents degrés, se retrouvent tout au long du développement :

En quoi la nature semi-structurée du corpus permet d'en automatiser le traitement ? Comment produire des informations normalisées et exploitables à partir d'un corpus textuel semi-structuré ? Comment rendre la recherche en humanités numériques réutilisable et encourager le partage de données entre projets de recherche ?

Les deux premières questions, d'orientation plutôt technique, forment la colonne vertébrale pour le mémoire ; elles lient deux aspects centraux : la nature du corpus et la manière dont sa structure permet toute la chaîne de traitement. Par « semi-structuré », j'entends que, à un niveau distant, toutes les entrées de catalogue suivent la même structure ; des séparateurs distinguent les différentes parties, et les informations sont souvent structurées de manière semblable pour chaque manuscrit vendu. Cela permet un traitement de « basse technologie » (*low-tech*) en évitant d'entraîner de lourds modèles de traitement du langage naturel (ce qui aboutirait à des solutions complexes, difficiles à maintenir et à faire évoluer et relativement opaques dans leur fonctionnement). À l'inverse, un corpus semi-structuré peut être traité en déduisant une « structure abstraite », que chaque entrée de catalogue partage. Il est alors possible de mettre en place des solutions techniques plus faciles, pour un résultat de qualité équivalente. Produire des « informations normalisées et exploitables » implique de traiter le corpus en cherchant des réponses à des questions de recherche précises – dans le cadre de mon stage, une question centrale a été de chercher à isoler les facteurs déterminant le prix d'un manuscrit.

Les deux dernières questions, au premier abord plus théoriques, me semblent centrales, notamment à la troisième partie de ce mémoire. Numérisation, traitement informatisé et diffusion sur le web ne sont pas des opérations neutres, mais un ensemble de « traductions » des documents originels. Ces processus comportent une part de choix conscients, qu'il s'agit de mettre en avant. Par exemple, on considère que la majorité des documents vendus ont pour titre l'auteur.ice du document. Cette personne n'est cependant pas toujours mentionnée, et des documents peuvent être nommés d'après un lieu, un évènement ou un thème (la Révolution française, par exemple). Ces « traductions » des catalogues sont relativement discrètes tout au long de la chaîne de traitement (où le format dominant est la `TEI`, qui garde une relation d'équivalence avec le texte). C'est lors

du passage au site web que ce processus de traduction devient plus évident, et, potentiellement, plus problématique. On y abandonne la référence au document originel (les catalogues numérisés ne sont pas accessibles en ligne par un.e utilisateur.ice), le catalogue n'est plus la manière privilégiée d'accéder aux items vendus... De plus, la construction d'un site web implique la conception d'une interface et, dans notre cas, la production d'une série de visualisations intégrées au site. Le passage au site web remet aussi en cause la hiérarchie habituelle entre ingénierie et recherche : la conception d'un site ne répond pas à une question scientifique, mais elle soulève ses propres questions. Loin d'être anodines, ces problématiques de design déterminent la construction et la réception des savoirs. Il est donc important, je pense, de problématiser ces questions de visualisation et de design.

Première partie

Du document numérisé au XML-TEI : nature du corpus, structure des documents et méthode de production des données

Chapitre 1

Le marché des manuscrits autographes au prisme des catalogues de vente

Ce chapitre présente l'objet d'étude du projet *MSS* : étudier le marché des manuscrits autographes du XIX$^{\text{ème}}$ s.. parisien à partir de ses catalogues de vente et étudier la construction du canon littéraire au prisme du marché du manuscrit.

1.1 Pourquoi étudier le marché des manuscrits autographes ?

Cette section porte sur l'intérêt scientifique des objets d'étude du projet (marché des manuscrits et étude de la construction du canon).

1.2 La structure du corpus : périodisation, producteurs des documents et classification

Ici est faite une présentation des documents traités dans le cadre du projet MSS. La présentation est à deux niveaux : au niveau du corpus et des catalogues. Ce chapitre s'appuie sur les mémoires effectués par d'ancien.ne.s stagiaires de *Katabase*, qui ont déjà beaucoup analysé la nature et les enjeux du corpus [1].

1. Lucie Rondeau du Noyer, *Encoder automatiquement des catalogues en XML-TEI. Principes, évaluation et application à la revue des autographes de la librairie Charavay*, Mémoire pour le diplôme de master "Technologies numériques appliquées à l'histoire", Paris, École nationale des Chartes, 2019, URL : `https://github.com/lairaines/M2TNAH` (visité le 13/06/2022) ; Caroline Corbières, *Du catalogue au fichier TEI. Création d'un workflow pour encoder automatiquement en XML-TEI des catalogues d'exposition.* Mémoire pour le diplôme de master "Technologies numériques appliquées à l'histoire", Paris, École nationale des Chartes, 2020, URL : `https://github.com/carolinecorbieres/Memoire_TNAH`

1.2.1 Le corpus de catalogues de vente de manuscrits

Ici est présenté le corpus : nature, quantité de documents (et d'entrées individuelles), dates, différentes classifications qui peuvent être faites (revues, catalogues de ventes aux enchères ou à prix fixes...).

1.2.2 Structure des catalogues

Ici sera présentée la structure des catalogues ; la structure de chaque page ne sera détaillée qu'à la partie suivante.

(visité le 13/06/2022) ; Juliette Janès, *Du catalogue papier au numérique. Une chaîne de traitement ouverte pour l'extraction d'information issue de documents structurés*, Mémoire pour le diplôme de master "Technologies numériques appliquées à l'histoire", Paris, École nationale des Chartes, 2021, URL : `https://github.com/Juliettejns/Memoire_TNAH` (visité le 13/06/2022).

Chapitre 2

Production des données : de l'OCR à la TEI

Cette partie s'attache autant à présenter le processus d'océrisation (qui est déjà bien établi et ne constitue pas le cœur de mon stage) que la structure des documents. Alors que le chapitre précédent s'intéresse aux catalogues dans leur ensemble, ici, on étudie le corpus au niveau de la page et de l'entrée individuelle. En effet, l'océrisation repose sur la segmentation, et donc sur l'établissement d'une structure « abstraite » d'une page (c'est-à-dire, d'un découpage de la page en zones).

2.1 Extraire le texte des imprimés

2.1.1 Comprendre la structure du document pour préparer l'édition numérique

Appréhender la structure de la page à l'aide de SegmOnto

La structure des catalogues est présentée au niveau de la page. L'ontologie SegmOnto [1] est utilisée, autant pour appréhender la structure de la page que pour exprimer cette structure de façon standardisée.

Description des entrées de catalogue : préparer l'édition TEI

Ici, la structure des catalogues est présentée au niveau de l'entrée, c'est à dire du lot mis en vente. C'est à autour de la structure des entrées qu'est construite l'édition XML-TEI. On s'intéresse à la structure des entrées individuelles à deux niveaux :

1. Kelly Christensen, Simon Gabay, Ariane Pinche et Jean-Baptiste Camps, « SegmOnto – A Controlled Vocabulary to Describe Historical Textual Sources », dans *Documents anciens et reconnaissance automatique des écritures manuscrites*, Paris : École nationale des Chartes, 2022.

— Au niveau intellectuel : quelles sont les différentes parties d'une entrée (titre, description du manuscrit, prix...).

— Au niveau « textuel » : quels sont les séparateurs, c'est à dire les éléments dans le texte qui permettent de séparer les pages de catalogue en entrées et les entrées en sous-éléments) correspondant à la structure intellectuelle décrite ci-dessus.

2.2 L'encodage des manuscrits en XML-TEI

2.2.1 Encoder les catalogues en TEI

Ici est présentée la représentation XML-TEI des catalogues de vente.

2.2.2 L'encodage en TEI : un processus sélectif qui réduit les significations du texte

Après une étape d'océrisation via *eScriptorium*, le texte extrait des PDF peut être exporté soit en texte brut, soit en XML Page ou Alto. Ces formats s'attachent à garder une relation entre le XML et le document numérisé (les zones de texte sont indiquées, chaque ligne est dans une balise...). Cependant, l'unité intellectuelle centrale à la suite du projet, ce n'est pas la page numérisée, mais l'entrée de catalogue. Un format plus complexe que le XML d'*eScriptorium* est donc nécessaire. Assez logiquement, la suite du projet s'appuie sur une traduction des catalogues en TEI. On s'intéresse autant à la structure des documents XML (quelles balises sont utilisées...) qu'à l'intérêt scientifique d'une édition numérique (balisage sémantique, possibilité de normaliser les informations grâce à des attributs).

L'édition numérique en XML-TEI des catalogues implique une certaine perte d'informations : l'intégralité des significations contenues dans les catalogues imprimés ne peut être traduite en TEI (la police, ou la qualité du papier, peuvent être documentés mais ne peuvent pas être reproduites). Ce genre de perte d'information a lieu, à différents degrés, dans la plupart des éditions TEI : ce format n'est pas un substitut des documents originels. Dans le projet *MSS / Katabase*, d'autres informations sont perdues : l'édition numérique n'est pas censée être une représentation exhaustive des catalogues. La TEI n'est pas utilisée comme un format de conservation, mais comme un format de traitement qui sera enrichi dans les différentes étapes. Afin de mesurer ce qui est conservé et ce qui est perdu du document originel, l'édition TEI sera analysée à la lumière de la « roue du texte » du philologue Patrick Sahle [2] qui modélise les significations plurielles d'un texte.

cf Smith+Blackwell (2012) : texte (édition critique) peut être rpr sous la forme d'un graphe RDF de citations

2. Patrick Sahle, « Digital modelling. Modelling the digital edition », dans *Medieval and modern manuscript studies in the digital age*, London/Cambridge, 2016, URL : https://dixit.uni-koeln.de/wp-content/uploads/2015/04/Camp1-Patrick_Sahle_-_Digital_Modelling.pdf, p. 11.

parler aussi de l'OHCO (renear_refining_1996) et des critiques qui lui ont été faites dès les débuts de la TEI

John Frow + Susan Pearce ?

Deuxième partie

Vers une étude des facteurs déterminant le prix des documents : alignement des entrées de catalogue et enrichissement sémantique avec *Wikidata*

Cette partie est construite à partir d'une question de recherche : comment produire des informations exploitables pour une étude économétrique à partir d'un corpus textuel semi-structuré ? Un des objectifs du projet est de faire l'étude des facteurs déterminant le prix d'un manuscrit. Cette étude nécessite de produire des données normalisées pour toutes les entrées de catalogues. Des informations sur les items vendus ont déjà été extraites et normalisées. Elles sont principalement quantitatives : prix des manuscrits, dimensions et nombre de pages, date de création. Il manque cependant des informations sur les auteur.ice.s de manuscrits ; le statut de ceux-ci doit probablement influencer le prix des manuscrits. De telles informations ne sont pas présentes dans les catalogues. Il est donc nécessaire de récupérer des données depuis des sources extérieures afin d'enrichir les catalogues : c'est cet enrichissement, qualifié de sémantique, qui permettra d'étudier la relation entre facteurs biographiques et prix d'un manuscrit. Pour réaliser cet enrichissement, il a été choisi d'associer les auteur.ice.s des manuscrits avec des entités dans une base de connaissances en ligne : *Wikidata*. Concrètement, dans une telle base de connaissances, une entité est représentée par un identifiant auquel sont liés d'autres données. Il s'agit donc de récupérer des identifiants depuis *Wikidata* pour construire une base de connaissances sur mesure à propos des auteur.ice.s de manuscrits via `SPARQL`.

Ce travail d'enrichissement a été fait en deux temps. La première étape, et la plus difficile, est l'alignement avec *Wikidata*. Cela demande d'extraire un ensemble d'informations à partir du nom de la personne et de la description de celle-ci. Parmi les informations extraites se trouvent : le nom, le prénom, le titre de noblesse d'une personne, ainsi que son occupation et ses dates de vie et de mort. À partir de ces informations, un algorithme lance plusieurs recherches sur *Wikidata* afin d'aligner les données des catalogues avec les bonnes entités. La deuxième étape, nettement plus simple, consiste à faire des requêtes `SPARQL` sur les identifiants *Wikidata* afin de récupérer des informations sur les auteur.ice.s des manuscrits. C'est cette étape qui constitue l'enrichissement sémantique à proprement parler. Pour finir, les fichiers `TEI` sont mis à jour pour comprendre les identifiants *Wikidata*. Ainsi, il est possible de faire le lien entre les entrées de catalogues dans des fichiers `XML` et les données issues de requêtes `SPARQL`, stockées dans un `JSON`.

L'alignement des auteur.ice.s de manuscrits avec des entités *Wikidata* soulève deux importantes questions d'ingénierie des données. (1) Comment s'appuyer sur la structure des catalogues pour développer un comportement uniforme basé sur de la détection de motifs pour un corpus complexe ? Par « détection de motifs », il faut comprendre l'utilisation de caractéristiques formelles du texte pour permettre l'extraction d'informations. (2) Comment utiliser le Web de données pour produire des données adaptées au projet et à la problématique de recherche centrale (une étude économétrique du corpus), mais aussi réutilisables à d'autres fins ?

Mais au delà de la question de recherche présentée en début de partie et de ces problématiques techniques, cette partie s'intéresse également aux méthodes utilisées en

humanités numériques, et cherche à comparer la méthode établie ici avec d'autres techniques plus répandues, mais aussi plus complexes et basées sur l'apprentissage machine, pour lier des textes avec des bases de connaissances. En comparant les méthodes utilisées, la possibilité de les adapter à d'autres corpus et les résultats obtenus, cette partie s'attache à étudier la possibilité de développer des méthodes « alternatives », basées sur la structure des documents traités, pour le traitement de données textuelles, tout en montrant comment les méthodes établies peuvent servir d'inspiration pour les possibilités d'exploitation du liage avec des entités *Wikidata*.

Chapitre 3

Questions introductives : pourquoi et comment s'aligner avec *Wikidata* ?

Cette section, introductive, répond à des questions évidentes mais essentielles : elles permettent de mettre au clair l'intérêt et les (multiples) difficultés dans l'alignement avec *Wikidata*.

3.1 Questions théoriques

3.1.1 Pourquoi s'aligner avec des entités externes ?

L'enrichissement des données depuis une base de connaissances a pour objectif principal de mieux comprendre les déterminants du prix des manuscrits sur le marché du XIX$^{\text{ème}}$ s.. Mais pourquoi passer par cet alignement ?

Pour étudier les déterminants du prix d'un manuscrit, il faut établir la relation entre le prix d'un manuscrit et un ensemble d'autres facteurs (qui a écrit un manuscrit, quelles sont ses dimensions, de quand date le document...). En d'autres termes, il faut étudier le comportement d'une variable en fonction d'autres variables. En économétrie, cette opération s'appelle le calcul de régressions linéaires. La variable étudiée (le prix) est dite la variable expliquée ; les facteurs déterminant sa valeur sont dites « variables explicatives » [1]. Cependant, cette opération est loin d'être anodine : il faut d'abord identifier les variables pertinentes, et ensuite trouver un moyen de les quantifier. Deux difficultés se présentent alors.

Premièrement, il faut pouvoir quantifier les variables expliquées pour calculer des régressions linéaires. Il est possible de leur assigner une valeur numéraire (ce qui est aisé pour les informations quantitatives des catalogues : la date de l'écriture d'un manuscrit, ses les dimensions). Une autre possibilité est de quantifier la présence ou non d'une variable :

1. *Régression linéaire*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire` (visité le 10/07/2022).

mention d'un.e destinataire ou du contenu d'un manuscrit. Cependant, ces approches quantitatives ne permettent pas de quantifier des informations complexes, comme la célébrité des auteur.ice.s, ou encore si un manuscrit porte sur un évènement historique ou biographique important (le manuscrit d'un texte célèbre, par exemple, pourrait avoir une valeur particulières). Ces informations sont parfois être présentes dans les catalogues ; elles peuvent aussi être connues des lecteur.ice.s d'aujourd'hui et des acheteurs et acheteuses de l'époque. Il n'existe cependant pas de méthodes faciles pour détecter ou quantifier la célébrité d'une personne, ou l'importance d'un sujet.

Une deuxième difficulté découle justement de la part d'implicite qu'il y a dans les catalogues. Les descriptions des items vendus sont brèves, et comprendre ce qui fait la valeur d'un manuscrit demande aux acheteur.euse.s d'avoir des références culturelles et historiques : celles-ci permettent d'identifier l'auteur.ice ou le sujet, et donc pour comprendre la valeur d'un manuscrit. Dans le cadre du projet *MSS / Katabase*, les entrées de catalogues sont traitées par une machine qui, en toute logique, ne dispose pas de ces références. La compréhension qualitative des entrées de catalogues n'est donc pas compatible avec l'approche par lecture distante qui permet l'analyse de très larges corpus textuels. Pour éviter de perdre ces informations qualitatives essentielles, il est donc nécessaire de trouver un moyen de quantifier le qualitatif.

En bref, la question est : comment faire la différence entre une lettre de La Rochefoucauld (3.1a), vendue 200 francs, et la deuxième (3.1b), vendue à 30 francs ? Le problème est un problème de lecture. Une observation de la description des lettres par un être humain comme par une machine peuvent identifier des éléments semblables dans les textes : les deux lettres sont écrites par des ducs ; l'une est une « Très-belle lettre » (3.1a), l'autre est une « Lettre intéressante » (3.1b). Bien que les lettres partagent des attributs, il y a une forte différence de prix entre les deux manuscrits. Un.e lecteur.ice peut trouver une raison à cette différence de prix : La Rochefoucauld et Madeleine de Scudéry n'ont pas le même statut que le duc de Villars. Un lecture humaine peut donc interpréter un prix et déterminer une valeur en s'appuyant sur ses connaissances. La lecture est qualitative et s'appuie sur de l'implicite, ce qui n'est pas possible pour une machine : formellement, rien ne distingue un nom d'un autre ; lorsqu'un programme « lit » un texte, il ne peut pas s'appuyer sur ses connaissances pour déterminer ce qu'un nom signifie, ce à quoi il fait référence.

Pour analyser efficacement la variable « prix », il faut pourtant pouvoir, dans une certaine mesure, comprendre les informations implicites et qualitatives contenues dans les catalogues. Le parti pris a donc été de construire le socle de connaissance qui manque à une machine, en s'alignant avec *Wikidata* et en s'en servant pour enrichir nos données. Le choix a été fait de ne s'aligner avec *Wikidata* que pour certaines parties des entrées de catalogue. Pour rappel, leur structure est visible en 3.1.

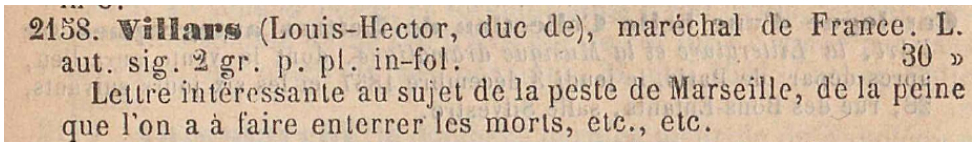
Les entrées de catalogue contiennent beaucoup d'informations qualitatives, qui pour-

```xml
<item n="287" xml:id="CAT_000126_e287">
    <num type="lot">287</num>
    <name type="author">Tascher de la Pagerie
    ↪ (Marie-Euphémie-Désirée)</name>
    <trait>
        <p>fille de Joseph Tascher qui passa à Saint-Domingue, et de
        ↪ Mlle de la Chevalerie; elle épousa M. de Renaudin, puis le
        ↪ marquis François de Beauharnais et contribua beaucoup à la
        ↪ fortune de la future impératrice Joséphine.</p>
    </trait>
    <desc xml:id="CAT_000126_e287_d1">
        Pièce originale;
        (<date when="1780">1780</date>),
        <measure type="length" unit="p" n="4">4 p.</measure>
        <measure type="format" unit="f"
        ↪ ana="#document_format_4">in-4.</measure>
    </desc>
    <measure commodity="currency" unit="FRF" quantity="15">15</measure>
    <note>Plaidoirie pour le dit Renaudin contre sa femme qu'on accusait
    ↪ d'avoir usé de l'influence de son parent, M. de Beauharnais,
    ↪ pour se faire épouser</note>
</item>
```

Code source 3.1 – Représentation `XML-TEI` d'une entrée de catalogue


24. **La Rochefoucauld** (François VI, duc de), le célèbre auteur des *Maximes*. L. aut. sig., à Mlle de Scudéry. Paris, 22 août... 2 gr. p. pl. et quart in-4. Cachets et soies. Très-belle lettre. 200 »

(a) Une lettre écrite par La Rochefoucauld vendue à 200 francs.


2158. **Villars** (Louis-Hector, duc de), maréchal de France. L. aut. sig. 2 gr. p. pl. in-fol. 30 »
Lettre intéressante au sujet de la peste de Marseille, de la peine que l'on a à faire enterrer les morts, etc., etc.

(b) Une lettre écrite par Louis-Hector Villars vendue à 30 francs.

FIGURE 3.1 – Deux exemples de lettres

raient avoir une influence sur le prix du manuscrit : dans l'exemple d'encodage présenté en 3.1, la description du contenu de la lettre dans le `tei:note` ; il est également souvent fait mention du ou de la destinataire. Cependant, l'alignement avec *Wikidata* n'a pas été fait avec l'intégralité des entrées, mais seulement le contenu du `tei:name`. Le `tei:desc` a déjà fait l'objet d'un grand travail de normalisation et d'extraction d'informations ; un alignement avec des sources externes n'aurait donc pas une très grande plus-value. L'élément `tei:note` contient souvent des informations intéressantes, puisque c'est là qu'est décrit le contenu d'un manuscrit. Cependant, cet élément n'est pas toujours présent ; son contenu est souvent écrit en langage naturel, non structuré, et contient des informations trop variées pour développer un traitement uniforme. Il est donc difficile de tirer parti de cet élément. Le `tei:name` et son `tei:trait` sont les éléments les plus régulièrement présents ; les informations qu'ils contiennent sont toujours les mêmes (nom d'une personne ou thème d'un manuscrit dans le `tei:name`, description du `tei:name` dans le `tei:trait`) ; enfin, ces deux éléments n'ont pas du tout été mis à profit dans le reste de la chaîne de traitement. Ils portent donc des informations qualitatives centrales pour produire des données exploitables dans une étude économétrique.

Le parti pris a donc été d'aligner avec des identifiants *Wikidata* les noms contenus dans les balises `tei:name` à l'aide des descriptions contenues dans les `tei:trait` ; à partir de cet alignement a été constituée une base de connaissances sur mesure, adaptée aux besoins du projet. Cela permet d'approximer une lecture « humaine » des items en vente : pour chaque auteur.ice, un certain nombre d'informations auront été récupérées pour mieux identifier la personne (ses occupations, son origine, ses dates de vie...). L'analyse du corpus s'appuie alors sur un bagage de connaissances qui permet d'appréhender par lecture distante l'importance d'une personne. Il devient alors envisageable de voir dans quelle mesure la mention d'une personne impacte le prix d'un manuscrit, et quels sont les facteurs biographiques déterminant la construction de la valeur. Pour revenir à l'exemple de La Rochefoucauld : à défaut de permettre de savoir qui il est, un alignement avec *Wikidata* permet de mieux situer son statut et sa place dans la culture française, en récupérant le nombre de ses publications ou encore les institutions dont il est membre.

3.1.2 Pourquoi utiliser la base de connaissances *Wikidata* ?

De nombreuses bases de connaissances sont accessibles en ligne ; celles-ci ont pour rôle de stocker et de diffuser des données structurées. Dans le paysage de la diffusion d'informations sur le Web, les bases de connaissances s'opposent dont à de nombreux projets qui diffusent du texte, comme l'encyclopédie *Wikidata*. Elles diffèrent également des *dumps* de données, c'est-à-dire à la diffusion de jeux de données publics sur le Web. Les *dumps* – tels qu'ils sont pratiqués par *Data.gouv*, structure développée par le gouvernement français – sont des jeux de données « inertes » : les données sont diffusées dans des fichiers qui ne seront donc pas mis à jour. Une base de connaissances donne accès à de l'information brute, directement exploitable ; les informations y sont accessibles individuellement depuis le Web (alors qu'elles sont contenues dans de lourds fichiers qui doivent être téléchargés dans le cas des « dumps »). Certaines de ces bases de connaissances ont été créées par des institutions publiques, comme *DataBnF*, créée par la Bibliothèque nationale de France. D'autres ont des sources universitaires, comme *DBPedia* ou communautaires, c'est-à-dire qu'elles sont enrichies par une communauté d'utilisateur.ice.s – qui peuvent être des acteurs institutionnels. Enfin, certaines de ces bases de connaissances sont spécialistes, comme le Thésaurus INHA, qui met à disposition du public les vocabulaires contrôlés développés par l'Institut national d'histoire de l'art ; d'autres, comme *DBPedia*, sont généralistes. Un paysage varié – brièvement présenté ici – de solutions existent donc pour récupérer des données. Parmi cette variété de sources, laquelle choisir ? Cette question fait l'objet de nombreuses études, notamment issues du domaine du traitement automatisé du langage [2]. Des algorithmes complexes ont été développés pour juger de la qualité des données et de leur adaptation aux corpus d'un projet de recherche. L'utilisation de tels algorithmes ne s'intègre pas dans les méthodes du projet *Katabase* : ils sont pensés pour être utilisés avec des méthodes d'apprentissage machine. Le projet *MSS / Katabase* évite, lorsque cela est possible d'utiliser de tels outils. Une chaîne de traitement pour l'alignement avec *Wikidata* a de plus déjà été élaborée ; il n'était pas pertinent de modifier toute la chaîne de traitement simplement pour déterminer si une meilleure base de connaissances pourrait être utilisée.

La sélection de la base de connaissances à utiliser a donc été faite à partir d'une évaluation des besoins du projet. D'un point de vue des données, il était nécessaire d'utiliser une base généraliste, disposant d'informations biographiques sur des personnes et de données pertinentes pour un corpus issu de la France métropolitaine du XIX$^{\text{ème}}$ s.. D'un point de vue technique, la base de connaissances devait disposer de deux outils. D'abord, l'alignement des `tei:name` avec des entités *Wikidata* devait être fait à l'aide de recherches en plein texte sur le moteur de recherche de la base ; pour automatiser ce processus,

2. Carmen Brando, Nathalie Abadie et Fontini Frontini, « Évaluation de la qualité des sources du web de données pour la résolution d'entités nommées », *Ingénierie des systèmes d'information*, 21–5 (28 déc. 2016), p. 31-54, DOI : 10.3166/isi.21.5-6.31-54.

la base de connaissances devait donc disposer d'une *Application Programming Interface* (« Interface de programmation d'application », ou API) qui permette une telle opération. Ensuite, la constitution d'une base de données sur les auteur.ice.s de manuscrits devait être faite à l'aide du langage de requêtes `SPARQL`, qui permet la récupération automatisée d'informations structurées sur une base de données en ligne. *Wikidata* répond à ces deux conditions. Mais cette base de donnée communautaire, développée par la *Wikimedia Foundation* (également à l'origine de *Wikipédia*) et rendue publique en 2012 présente de nombreux autres avantages. Contrairement à *DBPedia* (dont les données sont issues de l'extraction d'informations depuis *Wikipedia*) [3], les données de *Wikidata* sont directement produites par les contributeur.ice.s. Le projet est bicéphale, composé d'un côté de l'ajout d'informations à destination du public, et de l'autre de la construction d'une ontologie ; la particularité de *Wikidata* est que les deux aspects sont collaboratifs, alors qu'en général, la construction d'une ontologie est le fait d'ingénieur.e.s professionel.le.s [4]. En 2015, la communauté *Wikidata* représente 300000 personnes et 160 *bots* (des programmes informatiques qui ajoutent automatiquement des données sur *Wikidata*) [5]. Cette communauté est basée sur une forme de division du travail entre humain.e.s et machines : 85% des ajouts sont faits par des *bots*, mais ce sont des ajouts simples et automatisables (par exemple, l'ajout d'identifiants uniques pour lier les entités à d'autres sources de données). La part humaine des contributions est consacrée aux ajouts plus complexes, avec une minorité d'acteur.ice.s (30%) qui réalisent la très grande majorité des éditions, et sur de nombreux projets. À l'inverse, Les 70% restants ont une contribution bien plus faible, et centrée sur un ou quelques sujets de prédilection. L'immense majorité des éditions représente des ajouts d'informations, tandis que l'évolution de la structure de *Wikidata* repose sur une poignée d'utilisateur.ice.s très spécialisé.e.s [6]. Rien que par son envergure et la quantité de données disponibles, *Wikidata* est un candidat intéressant pour le projet d'alignement des `tei:name` avec une base de connaissances externe. Par ailleurs, sa nature généraliste s'adapte à la vaste période couverte par le corpus, ainsi qu'à la variété d'informations qui peuvent être contenues dans le `tei:name`. La nature communautaire du projet est elle aussi intéressante : *Wikidata* est enrichi par une communauté internationale, et de nombreuses données sont donc disponibles pour un corpus français ; des bases non-communautaires risquent toujours d'avoir un biais national dans les données disponibles.

Le modèle de données et les choix techniques font de *Wikidata* une source adaptée aux objectifs du projet *Katabase*. Tous les objets qui sont documentés par *Wikidata*

3. Claudia Müller-Birn, Benjamin Karran, Janette Lehmann et Markus Luczak-Rösch, « Peer-production system or collaborative ontology engineering effort : what is Wikidata ? », dans *Proceedings of the 11th International Symposium on Open Collaboration*, San Francisco California, 2015, p. 1-10, DOI : `10.1145/2788993.2789836`, p. 1.

4. *Ibid.*, p. 2-3.

5. *Ibid.*, p. 4.

6. *Ibid.*, p. 4, 8-9.

(appelés entités) peuvent être classés en deux types [7] :

— Les items, c'est-à-dire des données sur laquelle d'autres informations sont disponibles. Ceux-ci correspondent à des choses physiques ou à des concepts du monde réel, et peuvent être classifiés en deux catégories : les classes et les individus. Les classes sont des catégories génériques (« une personne ») tandis que les individus sont des membres de ces catégories (« Natalia Gontcharova »).

— Les propriétés, qui caractérisent les relations entre un ou plusieurs items. Par exemple, « occupation » permettrait de lier l'item « Natalia Gontcharova » à l'item « peintre ».

D'un point de vue technique, ce modèle est intéressant : il correspond à l'organisation des données propre au Web sémantique et à `SPARQL` [8] ; cela facilite l'utilisation ultérieure de ce langage de requêtes : la relation entre les données, qui est utilisée pour récupérer des informations avec `SPARQL`, est explicite ; il est donc facile de trouver comment structurer des requêtes. Là encore en concordance avec les modèles de données sémantiques, chaque item dispose d'un identifiant unique, d'un *Uniform Resource Identifier* (« Identifiant unique de ressource », ou URI) ; c'est ces identifiants qui sont utilisés dans `SPARQL` pour récupérer des données, ce qui facilite encore l'utilisation de ce langage [9]. Enfin, une dernière spécificité de cette base de connaissances est intéressante en vue de l'utilisation ultérieure de `SPARQL` : *Wikidata* a fait le choix de n'utiliser qu'un vocabulaire par catégorie de données. Comme nous le verrons plus tard, dans le Web sémantique, un vocabulaire est nécessaire pour définir les propriétés reliant plusieurs entités ; par exemple, un identifiant est relié à son nom grâce à la propriété `rdfs:label`, qui appartient au vocabulaire `rdfs`. Plusieurs vocabulaires ont souvent leurs propres manières de définir les mêmes relations, et les bases de connaissances ont tendance à utiliser plusieurs vocabulaires qui se chevauchent – c'est par exemple le cas de *DataBnF*. Cela rend le modèle de données utilisés par ces bases particulièrement complexe ; il est donc difficile de naviguer dedans, et encore plus difficile de récupérer des données via `SPARQL`. À l'inverse, la majorité des propriétés sont définies par l'ontologie de *Wikidata*, qui est très bien documenté sur le site ; cela rend la manipulation des données sur cette base de connaissances particulièrement aisée. Ainsi, *Wikidata* dispose d'un modèle de donnée totalement adapté à l'utilisation de langages de requêtes dédiées aux modèles de données sémantiques.

Dans *Wikidata*, tout item contient les mêmes types informations. Trois sont particulièrement intéressantes pour le processus d'alignement : le label, la description et les alias [10]. Un label permet de nommer un item ; par conséquent, il est possible de distinguer

7. Fredo Erxleben, Michael Günther, Markus Krötzsch, Julian Mendez et Denny Vrandečić, « Introducing Wikidata to the Linked Data Web », dans *The Semantic Web – ISWC 2014*, dir. Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz et Carole Goble, Series Title : Lecture Notes in Computer Science, Cham, 2014, t. 8796, p. 50-65, DOI : 10.1007/978-3-319-11964-9_4, p. 51-52.

8. *Ibid.*

9. *Ibid.*, p. 56.

10. *Ibid.*, p. 52.

l'entité (représentée par un identifiant normalisé) et sa représentation linguistique (le label). Cette distance est intéressante pour le projet *Katabase* : l'alignement des `tei:name` avec des entités *Wikidata* permet de retrouver un identifiant à partir du label qui lui est associé. Ce label est unique ; cependant, il existe de nombreux alias, qui sont des représentations linguistiques multilingues ou des variations du nom associé à une entité. Cela veut dire que, même si, dans un catalogue, un.e auteur.ice n'est pas nommé.e de façon canonique (en traduisant son nom, par exemple), il est toujours possible d'aligner le `tei:name` avec une entité *Wikidata* à l'aide des alias. La description, enfin, contient souvent la date de naissance et de décès ainsi que l'occupation d'une personne ; autant d'informations qui sont contenues dans le `tei:trait` et qui seront donc utilisées dans le processus d'alignement. De l'expérience que j'ai pu en faire, label, alias et description sont indexés par le moteur de recherche de *Wikidata* : cela veut dire que en recherchant ces informations sur l'API, les données des catalogues pourront être alignées avec les entités pertinentes.

Pour toutes ces raisons, la base de connaissances pour le processus d'alignement avec une source de données externe est *Wikidata*. Cette plateforme dispose à la fois des outils techniques nécessaires, d'un socle de données correspondant à l'univers de discours du corpus et d'une certaine facilité d'utilisation, un critère non négligeable pour garantir la réutilisation ultérieure des processus développés.

3.1.3 Quelle relation avec la résolution d'entités nommées d'entités nommées ?

En traitement automatisé du langage, le processus d'alignement d'éléments d'un texte avec une base de connaissances externe a un nom : c'est la résolution d'entités nommées (plus tard abrégée sous son acronyme anglo-saxon, *Named Entity Linking* (NEL)). Pour reprendre la définition proposée par Brando *et al.* : « La tâche dite de résolution d'entités nommées [...] consiste à affecter automatiquement la bonne référence, l'identifiant de la ressource correspondante dans une base de connaissances ou bien NIL (l'absence de valeur), à une entité nommée préalablement identifiée dans un texte et étiquetée. » [11]. Dans un texte, ce processus arrive après l'identification d'entités, c'est-à-dire d'objets uniques et signifiants pour une problématique de recherche : personnes, lieux, mais aussi parfois concepts. Il consiste à aligner les entités du texte avec celles présentes sur une base de connaissances externe. Deux objectifs primaires peuvent être identifiés [12]. Le premier est la désambiguïsation des entités dans un texte, ce qui permet à une machine de ne pas confondre deux entités homonymes (Alexandre Dumas père et fils, par exemple). Cette étape est très importante pour le traitement automatisé d'un texte : si un.e lecteur.ice hu-

11. C. Brando, N. Abadie et Fontini Frontini, « Évaluation de la qualité des sources du web de données pour la résolution d'entités nommées »…, p. 31.

12. *Ibid.*, p. 32.

main.e est capable de distinguer les deux Alexandre Dumas en fonction de sa lecture et de ses références, cette distinction est impossible par une machine. La reconnaissance d'entités nommées permet alors d'expliciter l'implicite, ce qui est également l'un des objectifs ayant mené à l'alignement des auteur.ice.s de manuscrits avec des entités de *Wikidata*. Le deuxième objectif est « l'enrichissement sémantique », qui repose sur la récupération d'informations depuis une source externe pour permettre une meilleure compréhension et analyse du texte par des méthodes computationnelles. Comme cela a été dit plus haut, c'est exactement à cette fin qu'est mené l'alignement des `tei:name` avec *Wikidata*. Pourquoi alors ne pas directement parler d'un processus de résolution d'entités nommées ?

Tout d'abord, l'alignement avec *Wikidata* n'était pas censé prendre une telle ampleur à ses débuts. Il avait initialement été prévu de réaliser cet alignement « à la main », ce qui, pour un corpus de près de 100000 entrées, aurait été extrêmement chronophage. C'est avant tout pour répondre à ce problème purement pratique qu'ont été développées les méthodes pour automatiser l'alignement qui sont présentées ici. Étant donné que les entrées sont riches d'informations, il est apparu que ce processus pouvait être réalisé automatiquement, avec bien sûr un certain degré d'erreur. En choisissant d'automatiser l'alignement, les méthodes d'extraction d'informations des catalogues et de recherches sur l'API se sont progressivement complexifiées afin de mieux s'adapter au corpus. L'alignement avec *Wikidata* a donc progressivement été développé, jusqu'à devenir un processus analogue à la résolution d'entités nommées.

Ensuite et surtout, la résolution d'entités nommées « traditionnelle » implique des méthodes en tout point différentes de celles qui sont présentées ici. Le NEL repose très fortement sur l'apprentissage machine. Il est d'abord utilisé pour identifier les entités dans le texte [13] ; ensuite, il sert à mener l'alignement des entités avec la base de connaissances elle-même. Pour ce processus, de nombreux outils ont été développés, comme `REDEN` [14] ou `DBpediaSpotlight` [15], conçu au sein du projet *DBPedia* pour la résolution d'entités en utilisant cette base de connaissances. Ces outils fonctionnent, d'une certaine manière, à l'inverse du processus mis en place ici. Un ensemble de références potentielles récupérées d'une base de connaissances à l'aide de requêtes `SPARQL` [16] ; ensuite, les entités du texte sont liées à une liste de candidats potentiels avec lesquels ils pourraient être alignés, issus des références extraites au début du processus ; enfin, l'entité est résolue en y assignant un item parmi la liste de candidats [17], À l'inverse, le processus mis en place ici consiste

13. *Ibid.*

14. C. Brando, Francesca Frontini et Jean-Gabriel Ganascia, « REDEN : Named Entity Linking in Digital Literary Editions Using Linked Data Sets », *Complex Systems Informatics and Modeling Quarterly–*7 (29 juil. 2016), p. 60-80, DOI : 10.7250/csimq.2016-7.04.

15. Pablo N. Mendes, Max Jakob, Andrés García-Silva et Christian Bizer, « DBpedia spotlight : shedding light on the web of documents », dans *Proceedings of the 7th International Conference on Semantic Systems - I-Semantics '11*, Graz, Austria, 2011, p. 1-8, DOI : 10.1145/2063518.2063519.

16. C. Brando, N. Abadie et Fontini Frontini, « Évaluation de la qualité des sources du web de données pour la résolution d'entités nommées »…, p. 35.

17. *Ibid.*, p. 33.

d'abord à extraire des données des textes, ensuite d'interagir avec un moteur de recherche pour sélectionner une entité parmi toutes celles disponibles sur la base de connaissances, et enfin de mener des requêtes `SPARQL` pour récupérer des données supplémentaires sur les items liés. Dans le détail, les méthodes utilisées pour la résolution d'entités sont également très différentes : elles se basent, de manière générale, sur une tentative d'aligner l'entité du texte avec une entité sur la base de connaissances par l'analyse de graphes [18]. Pour ce faire, les entités de la base sont représentées sous forme de graphes qui sont analysés pour trouver, au sein du graphe, l'entité pertinente ; l'apprentissage machine permet à un outil « d'apprendre » à reproduire tout seul ce processus d'analyse du graphe, afin d'aligner d'autres entités avec leur équivalent sur une base de connaissances. Dans le processus décrit ici, il n'y a ni analyse de graphes, ni apprentissage machine : l'alignement se base sur la détection de motifs à l'aide d'expressions régulières. Des informations sont détectées dans les catalogues, extraites et si besoin retraitées, avant d'être utilisées pour faire des recherches sur *Wikidata* afin d'aligner un.e auteur.ice avec une entité de cette base de connaissances. Le processus décrit ici privilégie donc une compréhension approfondie du corpus, de sa structure et de son vocabulaire, ainsi qu'une bonne connaissance du fonctionnement du moteur de recherche de *Wikidata*. Si les méthodes sont radicalement différentes, les possibilités ouvertes sont équivalentes. Les scores de performance des deux méthodes sont, comme nous le verrons, relativement proches, ce qui ouvre à la possibilité de chercher à développer des méthodes alternatives pour la résolution d'entités nommées.

3.2 Questions de méthode : comment faire le lien entre un corpus historique et une base de connaissances en ligne ?

3.2.1 Présentation générale de l'algorithme

Construire un jeu de données issu de `SPARQL` à partir de la manière dont une personne est nommée et décrite au XIX$^{\text{ème}}$ s. n'est pas une opération anodine. La chaîne de traitement est donc assez complexe, comme le montre le schéma 3.2. Cette chaîne de traitement peut être séparée en trois étapes.

Étape 1 – Extraction et structuration de données

Premièrement, il s'agit d'aligner les entrées de catalogue avec des entités *Wikidata*. Cette première étape repose avant tout sur l'extraction et la traduction des données depuis les éléments `tei:name` et `tei:trait`. Ce processus d'extraction permet de récupérer toutes les données pertinentes pour chaque entrée de catalogue et de les stocker dans un

18. C. Brando, Francesca Frontini et J.G. Ganascia, « REDEN… », p. 63.

dictionnaire structuré. Comme on le verra, la nature « semi-structurée » des entrées (ainsi qu'une bonne connaissance du corpus) permet de d'automatiser le processus d'extraction et de traduction des données par détection de motifs, sans avoir à passer par l'apprentissage machine : étant donné que les mêmes types d'informations sont toujours présentes et que les entrées suivent des modèles relativement proches, il est possible de s'appuyer sur la structure des entrées pour identifier les informations pertinentes. L'extraction de données repose donc sur de la détection de motifs à l'aide d'expressions régulières : des récurrences sont repérées dans le texte et utilisées pour distinguer différentes informations (nom, prénom, titre de noblesse...). Pour appuyer l'usage d'expressions régulières par une méthode plus « qualitative » et précise, certains termes particuliers sont extraits et éventuellement traduits à l'aide de tables de conversion (c'est-à-dire de dictionnaires qui associent à un terme dans le texte une version normalisée).

Étape 2 – Récupération d'identifiants *Wikidata* via des recherches en plein texte à l'aide d'une API

Une fois les données du `tei:name` et du `tei:trait` structurées en dictionnaire, elles sont utilisées pour lancer plusieurs recherches en plein texte sur le moteur de recherche de *Wikidata*. Ces recherches sont faites automatiquement grâce à l'API de *Wikidata*. Pour maximiser les chances d'obtenir un identifiant valide, un algorithme a été conçu pour lancer plusieurs recherches à partir de chaque entrée. La première recherche met bout-à-bout toutes les valeurs disponibles dans le dictionnaire. Ensuite, en fonction des paramètres de recherche disponibles dans le dictionnaire, différentes autres recherches sont lancées. Cet algorithme a été élaboré en menant de nombreux tests pour maximiser le taux de réussite, calculé sous la forme d'un score F1.

Étape 3 – constitution d'un jeu de données à l'aide de `SPARQL`

Si l'étape d'alignement avec *Wikidata* est la plus complexe, elle n'est qu'une étape préparatoire vers la constitution du jeu de données et à l'enrichissement sémantique. En fait, résoudre les entités est ce qui rend possible l'enrichissement en tant que tel : en lançant une requête `SPARQL` sur tous les identifiants obtenus, il est possible de récupérer des informations depuis *Wikidata* et donc de construire le jeu de données définitif. Pour cette étape, le processus est plus simple : les identifiants récupérés à la fin de l'étape précédente sont dédoublonnés (pour éviter de faire plusieurs fois la même requête) ; ensuite une requête `SPARQL` est initalisée et faite chacun des identifiants. Les résultats sont traduits depuis les formats `JSON` ou `XML` retournés par `SPARQL` sous forme de `JSON` plus simple, et donc plus aisément manipulable. Le jeu de données est enregistré dans un fichier. Pour finir, les identifiants *Wikidata* sont réinjectés aux catalogues `TEI`, afin de pouvoir faire le lien entre les documents et le jeu de données qui a été construit.

Cette chaîne de traitement étant lancée sur plus de 80000 entrées de catalogues, le temps d'exécution est très long et même des petites améliorations de performance peuvent avoir un grand impact ; dans sa version initiale, le script demandait des performances particulièrement élevées, et ne fonctionnait pas sur un ordinateur de bureau. La chaîne de traitement a donc été reprise en plusieurs points afin d'être optimisée, de fonctionner plus vite en étant moins coûteuse en ressources.

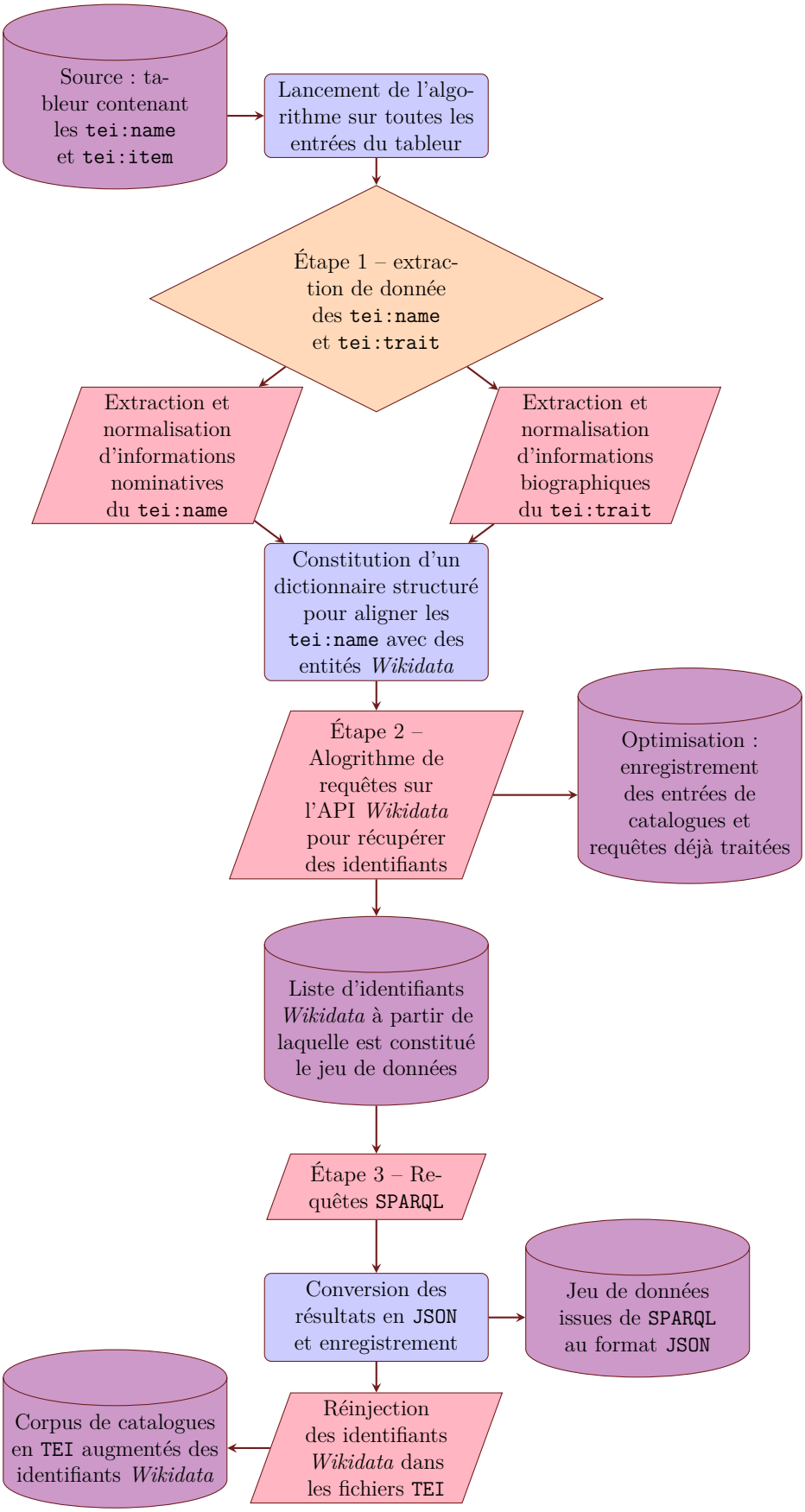


FIGURE 3.2 –
Présentation générale de l'algorithme d'enrichissement de données à l'aide de *Wikidata*

3.2.2 Comment traduire des descriptions textuelles datant du XIX^{ème} s. en chaînes de caractères pouvant retourner un résultat sur *Wikidata* ?

La principale difficulté de cette chaîne de traitement porte sur la récupération d'identifiants *Wikidata* à l'aide de recherches en plein texte. Le script prend en entrée un nom et sa description – tels qu'ils figurent dans des catalogues datant majoritairement du XIX^{ème} s.. La difficulté, au delà de la détection et de l'extraction d'informations, est de traduire ces informations pour qu'elles permettent de trouver des résultats pertinents sur *Wikidata*. Ce problème est autant linguistique de technique. Une personne ou une chose est nommée ou décrite d'une certaine manière dans un catalogue de vente ancien. Il n'y a aucune garantie que cette caractérisation corresponde à celle faite par *Wikidata* : l'orthographe des noms évoluent, tout comme la manière de nommer certains métiers. À ces évolutions orthographiques s'ajoutent des évolutions intellectuelles : les titres de noblesse sont un marqueur plus important au XIX^{ème} s. français que dans un XXI^{ème} s. mondialisé. Une personne n'est que rarement décrite par son titre dans *Wikidata*.

Le problème de la traduction des noms

Il existe bien sûr des cas simples, comme l'exemple 3.2 : en extrayant le contenu du `tei:name`et en traduisant le « roi » issu du `tei:trait`, la chaîne de caractère obtenue est « Henri IV king ». En recherchant cette chaîne de caractère sur *Wikidata*, le premier résultat obtenu est correct. Cependant, de nombreux cas sont plus complexes, surtout lorsque l'auteur.ice du manuscrit est moins célèbre. L'exemple 3.3 est éclairant : dans le catalogue, la personne est nomée « Bruno Daru » ; sur *Wikidata*, le nom de la personne est « Pierre Daru », et son nom complet « Pierre Antoine Noël Bruno Daru ». Si la recherche en plein texte est faite avec les mêmes paramètres que pour l'exemple précédent (nom de la personne et titre de noblesse), le premier résultat obtenu n'est pas le bon : c'est un renvoi à un article de *l'Encyclopedia Britannica* datant de 1911. C'est en cherchant seulement le nom est le prénom que *Wikidata* retourne un résultat pertinent. Deux choses sont à retenir de cet exemple : dans les catalogues, le prénom d'une personne correspond en fait souvent à son deuxième ou troisième prénom ; ensuite, le titre de noblesse est un critère plus fréquemment mentionné dans les catalogues que dans *Wikidata*. Cela s'explique assez aisément : le XIX^{ème} s. connaît une alternance de régimes politiques (royauté, empire, république) où la noblesse n'a pas encore perdu son pouvoir. La probabilité qu'un titre de noblesse soit mentionné sur *Wikidata* diminue lorsqu'un titre est peu important ; dans les catalogues, cependant, même les titres les moins importants sont régulièrement mentionnés. Par conséquent, seuls les titres les plus importants seront extraits pour faire des recherches sur *Wikidata*.

Dans le cas de noms de personnes étrangères, la situation peut être plus complexe

```xml
<item n="134" xml:id="CAT_000233_e134">
    <!-- ... -->
    <name type="author">Henri IV</name>
    <trait>
        <p>roi de France.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 3.2 – Un cas simple : Henri IV roi de France

```xml
<item n="98" xml:id="CAT_000082_e98">
    <!-- ... -->
    <name type="author">Daru (Bruno, comte)</name>
    <trait>
        <p>célèbre ministre de Napoléon Ier, historien de Venise, de
        ↪ l'Acad. fr., né à Montpellier</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 3.3 – Un cas plus complexe : Pierre Antoine Noël Bruno Daru

encore. L'exemple 3.4 combine différentes difficultés.

— D'abord, la personne est étrangère ; dans les catalogues, les noms sont systématiquement francisés – « Albert-Venceslas-Eusèbe » dans le catalogue, « Albrecht Wenzel Eusebius » en langue originelle. Se pose donc la question de si le nom doit être traduit, et si oui comment ?

— Ensuite, comme l'indique la présence de « dit » dans le `tei:name`, il est mentionné un nom de naissance (« de Waldstein ») et un nom d'usage (« Wallenstein »). Idéalement, il faudrait choisir entre l'un ou l'autre, plutôt que de rechercher « Waldstein Wallenstein » sur *Wikidata*, ce qui risque d'augmenter le bruit.

Notre approche s'appuyant sur la structure du texte, le deuxième point peut être réglé : le nom d'usage est écrit au début, et le nom de naissance entre parenthèses (c'est également le cas des noms de personnes nobles, par exemple). Il est donc possible de choisir l'un ou l'autre nom. Le premier point est plus problématique : si la traduction du nom serait envisageable en théorie, celle-ci est difficilement compatible avec une approche basée sur la détection de motifs dans le texte : un prénom est repérable comme étant un motif (trois noms séparés par des tirets) ; cependant, il est impossible de le traduire automatiquement (ce qui demanderait de connaître la langue dans laquelle un prénom doit être traduit). C'est ici que les informations contenues dans le `tei:trait` prennent leur

```
<item n="5518" xml:id="CAT_000401_e5518">
    <!-- ... -->
    <name type="author">Wallenstein (Albert-Venceslas-Eusèbe de
    ↪ Waldstein dit)</name>
    <trait>
        <p>duc de Friedland, célèbre général de la guerre de Trente ans.
        ↪ Assassiné en 1634.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 3.4 – Le problème des noms de personnes étrangères

importance : lorsqu'il y a des défaillances dans les informations nominatives, des données biographiques permettent de diminuer le risque d'erreurs. Dans cet exemple, rechercher « Albert-Venceslas-Eusèbe Waldstein » ne retourne aucun résultat, de même que rechercher « Albert-Venceslas-Eusèbe Wallenstein ». Cependant, le bon résultat est obtenu en recherchant « Wallenstein 1634 ». Une difficulté supplémentaire vient avec ce type de cas : différents paramètres de recherche (nom, prénom...) ont un impact différent dans l'obtention du bon résultat en fonction des personnes sur qui la requête est faite. Dans ce cas, rechercher le nom d'usage et la date de naissance retourne un résultat valide, ce qui n'est pas toujours le cas. Pour contourner ce problème, trois solutions ont été mises en place : d'abord, ce types de requêtes a été fait « à la main », de façon non-automatique, pour de nombreuses entrées différentes afin de déterminer la meilleure combinaison de paramètres ; ensuite, des tests qui permettent de mesurer l'influence de chaque paramètre de recherche dans l'obtention du résultat ont été réalisés ; enfin, l'algorithme final lance successivement différentes requêtes avec différents paramètres afin de maximiser la probabilité d'obtenir un résultat valide. Nous reviendrons plus en détail sur les deux derniers points.

L'extraction d'informations biographiques : une autre difficulté

Cependant, le problème ne s'arrête pas qu'aux noms. Dans un exemple ; précédent, le titre de noblesse influençait l'obtention d'un résultat valide. De nombreuses autres informations biographiques pourraient, au premier abord, permettre d'obtenir le bon résultat. C'est souvent le cas, puisque extraire le métier ou la fonction d'une personne permet de supprimer les faux positifs retournés par l'API. C'est par exemple le cas dans l'exemple 3.5. En cherchant uniquement le nom et le prénom (« Hans Bulow »), le premier résultat retourné renvoie à un journaliste suédois. Extraire le mot « pianiste » `tei:trait` et le traduit en anglais permet d'obtenir le bon résultat.

L'extraction d'informations biographiques et leur utilisation dans des requêtes est

```
<item n="136" xml:id="CAT_000189_e136">
    <!-- .. -->
    <name type="author">Bulow (Hans)</name>
    <trait>
        <p>le célèbre pianiste.</p>
    </trait>
</item>
```

Code source 3.5 – Un exemple où l'extraction du métier permet l'obtention du bon résultat

donc pertinent. Cependant, extraire trop d'informations conduit à lancer des requêtes qui ne renvoient aucun résultat. Dans les exemples 3.6 et 3.7, extraire et traduire des fonctions conduit à lancer les requêtes « John Okey colonel » et « Jean Bouhier president » qui ne retournent aucun résultat, ou des résultats qui ne sont pas valides. Cependant, dans les deux cas, si une requête est lancée sans le métier, un résultat correct est obtenu. Les raisons pour lesquelles des résultats erronés sont retournés ne sont cependant pas les mêmes, et il est intéressant de mieux observer les requêtes lancées et les résultats obtenus. Dans le premier exemple (3.6), le terme mis en avant dans le `tei:trait` (« colonel ») n'est pas celui avec lequel la personne est décrite sur *Wikidata* (où John Okey est décrit comme étant un homme politique). Cela met en avant un problème relatif au changement de regard sur des personnalités : dans un contexte, la personne est décrite comme une figure militaire, dans l'autre comme une figure politique. Le deuxième exemple (3.7) pose un problème plus technique. Il y a en fait une erreur dans la requête qui ne retourne pas de résultat (« Jean Bouhier president ») : un.e président.e de parlement n'est en général pas décrite comme « président ». Cependant, en extrayant des données uniquement par détection de motifs, il est possible de repérer et traduire un terme générique comme « président ». Extraire le complément « Parlement de Dijon » du `tei:trait` n'est cependant pas possible (cela impliquerait d'étudier la grammaire de la phrase, pour mettre en avant la relation entre « président » et « Parlement de Dijon »). Au vu de la taille et de la variété du jeu de données, il est impossible de traiter au cas par cas des entrées, ou préciser suffisamment la détection de motifs pour pouvoir résoudre ce genre de difficultés.

De situations comme les exemples 3.6 et 3.7, il faut donc retenir que l'extraction d'informations vient nécessairement avec un risque d'erreur. Le parti pris a donc été de ne pas repérer les métiers et autres termes très spécifiques, comme les grades militaires et les titres de noblesse peu élevés : ils ne retournent pas de résultats sur le moteur de recherche. Ensuite, plus des requêtes sont précises, plus elles risquent de retourner du silence (c'est-à-dire, de ne pas donner de réponse) ; cependant, si un résultat est obtenu, il est plus probable que ce résultat soit correct. Une fois l'extraction d'informations faite, l'algorithme de résolution d'entités avec *Wikidata* a donc été conçu en suivant un principe soustractif : les premières recherches sont faites avec un maximum de paramètres ; si aucun

```xml
<item n="152" xml:id="CAT_000189_e152">
    <!-- ... -->
    <name type="author">Okey (John)</name>
    <trait>
        <p>colonel anglais, un des lieutenants de Cromwell.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 3.6 – Quand l'extraction d'un métier conduit à des requêtes trop spécifiques

```xml
<item n="5430" xml:id="CAT_000401_e5430">
    <!-- ... -->
    <name type="author">Bouhier (Jean)</name>
    <trait>
        <p>président au Parlement de Dijon, membre de l'Académie
        ↪ française.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 3.7 – Le cas des métiers dont l'extraction est problématique

résultat n'est obtenu, des paramètres sont enlevés pour que l'API retourne un plus grand nombre de résultats. Enfin, ces deux exemples montrent qu'il n'est pas possible d'extraire et de traduire des informations sans prendre en compte ce qui sera pertinent pour le moteur de recherche de *Wikidata*. Il ne s'agit donc pas seulement d'extraire des informations, mais aussi de s'adapter à ce moteur de recherche pour augmenter la probabilité d'obtenir un résultat valide.

3.2.3 Comment négocier avec le moteur de recherche de *Wikidata* ?

Comme cela commence à apparaître, l'extraction d'informations, lorsqu'elle vise à interagir avec des données externes, vient avec des difficultés supplémentaires. Il ne faut pas seulement extraire les informations ; leur extraction et structuration doivent permettre de lancer des recherches en plein texte, et donc de minimiser le bruit (les informations non pertinentes) et le silence (l'absence d'informations) de la part du moteur de recherche. Il faut donc traduire les informations extraites pour qu'elles correspondent au vocabulaire utilisé par *Wikidata*. Cette opération n'est pas anodine : si les catalogues de vente fonctionnent avec leurs propres catégories, le même peut être dit de *Wikidata* : certains types de données sont plus souvent référencées que d'autres et *Wikidata* utilise

un vocabulaire qui lui est propre. Pour bien mener ce processus de traduction et de structuration de l'information, il est nécessaire de bien connaître le fonctionnement de ce moteur de recherche pour mieux s'y adapter.

Comme cela a été dit, l'alignement avec *Wikidata* passe par l'utilisation de l'API mise en point par l'institution afin de lancer automatiquement des recherches en plein texte ; l'objectif est que le premier résultat retourné par le moteur de recherche soit le bon. La première chose à remarquer est que, contrairement à un moteur de recherche généraliste (comme *Google, QWant...*), ce moteur utilisé par *Wikidata* n'est pas compatible avec des requêtes approximatives. L'exemple 3.8 est pertinent à ce égard [19]. Dans de nombreuses entrées, comme c'est le cas ici, les fonctions d'une personne ayant participé à la révolution sont présentées de façon précise : Marc David Alba Lasource est décrit comme étant un « conventionnel girondin ». Cette mention, régulièrement présente dans les catalogues, pourrait être relevée en tant que telle. Cependant, lancer la recherche « Lasource conventionnel » ne retourne aucun résultat. Si la même recherche est lancée sur un moteur de recherche généraliste (ici, *QWant*), la page *Wikipedia* de Lasource fait partie des premiers résultats [20]. Cette différence dans les données retournées par les moteurs de recherche a deux explications : un moteur de recherche généraliste recherche les occurrences de mots, non seulement dans le titre de la page, mais aussi dans le corps du texte. Si le mot « conventionnel » est absent du titre, il est certainement à plusieurs reprises dans une notice biographique de type *Wikipedia*. *Wikidata* ne contenant que des données, et pas de texte en tant que tel, l'indexation du corps du texte par le moteur de recherche interne à *Wikidata* n'est pas possible. Ensuite, la plupart des moteurs de recherche généralistes utilisent des méthodes de traitement du langage afin de simplifier la requête lancée par l'utilisateur.ice : les mots recherchés sont simplifiés, le moteur de recherche associe les termes recherchés avec d'autres termes « cooccurents », c'est-à-dire fréquemment utilisés ensemble [21]. Dans le cas du moteur de recherche de *Wikidata*, la requête de l'utilisateur.ice ne semble pas être retraitée : des signes de ponctuation ou des fautes de frappes influencent l'obtention d'un résultat, de même que l'usage de termes inadaptés.

19. Dans cet exemple, le prénom, « M.-D.-A. », n'est pas pris en compte pour se concentrer sur l'utilisation d'informations biographiques dans le `tei:trait`.

20. Le 29/07/2022, c'est le troisième résultat ; le premier correspond à une vente aux enchères d'archives du conventionnel. Les moteurs de recherche pouvant être mis à jour régulièrement, il est possible que l'ordre des résultats change

21. *Moteur de recherche*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://fr.wikipedia.org/wiki/Moteur_de_recherche` (visité le 17/07/2022). Pour des analyses plus détaillées sur la construction d'ensemble de termes coocurrents via le développement de vecteurs de mots, voir Tomas Mikolov, Kai Chen, Greg Corrado et Jeffrey Dean, « Efficient Estimation of Word Representations in Vector Space » (, 2013), Publisher : arXiv Version Number : 3, DOI : `10.48550/ARXIV.1301.3781` ; pour un article technique détaillant la classification et la sélection de résultats pertinents par apprentissage profond, voir Paul Covington, Jay Adams et Erme Sargin, « Deep Neural Networks for YouTube Recommendations », dans *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, 2016, URL : `https://research.google/pubs/pub45530/` (visité le 10/06/2022)

```xml
<item n="140" xml:id="CAT_000197_e140">
    <!-- ... -->
    <name type="author">Lasource (M.-D.-A.)</name>
    <trait>
        <p>célèbre conventionnel girondin, né près de Montpellier en
        ↪ 1762, guillotiné en 1793.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 3.8 – Le problème de l'approximation et de la traduction : Lasource, conventionnel

Pour faire face à la « rigidité » relative du moteur de recherche de *Wikidata*, il est donc nécéssaire de préparer ses données au moment de leur extraction. En prenant le même exemple (3.8), un résultat correct peut être obtenu en remplaçant « conventionnel » par « politician » [22], pour rechercher sur *Wikidata* « Lasource politician ». Ici, la traduction de « conventionnel » en « politician » est d'autant plus intéressante que la date de naissance dans le catalogue (1762) ne correspond pas à celle indiquée sur *Wikidata* (1763). Dans un cas comme celui-ci, où certaines données sont incorrectes, il est important d'extraire un maximum d'informations pour que, si certaines requêtes ne rapportent pas de résultats, pouvoir en faire d'autres avec différents paramètres.

En conclusion, il faut retenir que le moteur de recherche de *Wikidata* n'admet pas d'erreurs, ni de requêtes partiellement erronées (dans l'exemple 3.8, où la date de naissance soit correcte, mais pas la date de décès) ; il ne prend pas non plus en compte la synonymie, ce qui veut dire qu'il n'améliore pas la requête lancée par un.e utilisateur.ice. Cela signifie que les termes utilisés dans une requête doivent être adaptés à ceux que *Wikidata* utilise. Les termes spécifiques utilisés dans les catalogues (« conventionnel »), mais aussi de nombreux titres militaires et de noblesse peu élevés (« capitaine », « marquis ») sont relativement rarement présents sur *Wikidata*. Lorsque les requêtes sont lancées, de tels termes sont donc abandonnés et parfois remplacés par des termes plus génériques : par exemple, « capitaine » est remplacé par « military », traduction anglaise de « militaire ». De même, des termes principalement en usage dans la langue française, comme « conventionnel » sont moins efficaces pour lancer des recherches.

3.3 Une approche prédictive

L'alignement avec *Wikidata* et la résolution d'entités n'est donc pas une opération anodine : les données contenues dans les catalogues sont variées, autant par leur structure

22. « Personnalité politique »

```
1        <name type="author">Verneuil (Charlotte Séguier duchesse
         ↪  de)</name>
```

Code source 3.9 – Peut-on identifier les différents éléments d'une phrase par détection de motifs ?

que par les informations qu'elles contiennent ; il peut être difficile à faire la traduction de données du XIX^{ème} s. en chaînes de caractères pouvant retourner des réponses valides sur *Wikidata* ; enfin, le l'alignement repose sur une bonne connaissance du moteur de recherche de *Wikidata*.

De plus, la technique utilisée dans l'extraction de données, reposant sur la détection de motifs à l'aide de expressions régulières et de tables de conversion, est une technique qui vient avec un certain nombre d'incertitudes. Avec ce genre de techniques, il est impossible de « comprendre » ce qu'un élément signifie. Dans l'exemple 3.9, formellement, rien ne sépare le nom propre de la duchesse (« Séguier ») du nom de son duché (« Verneuil »). En s'appuyant sur une connaissance de la structure répétitive des entrées, il est uniquement possible de supposer que le nom entre parenthèses est un nom propre, tandis que le nom hors des parenthèses correspond au nom du duché. En bref, les méthodes de détection de motifs utilisées, peuvent uniquement inférer le sens d'un mot par rapport à sa position dans une phrase. Si cette technique implique une certaine incertitude, elle est cependant particulièrement adaptée à un corpus semi-structuré, comme c'est le cas des catalogues de vente de manuscrits, et à l'opération d'alignement avec *Wikidata*. Comme cela est expliqué plus bas, au fond, il n'est pas tellement important de distinguer le sens des différentes informations : ce qui compte, c'est que l'extraction et la structuration des informations permettte de construire des chaînes de caractères à rechercher sur *Wikidata*. Identifier la fonction d'un mot n'est donc ici qu'un moyen – contrairement à de l'analyse lexicale, où la fonction des mots dans une phrase est signifiante –. En effet, repérer le rôle que tiennent les termes extraits (métier, prénom...) permet de mieux construire la chaîne de caractère recherchée sur *Wikidata*, en pouvant filtrer certaines informations (retirer les dates de vie et de mort, par exemple) pour construire différentes requêtes à partir de la même entrée de catalogue.

Étant donnée cette quantité d'incertitudes, l'approche suivie dans l'alignement avec *Wikidata* peut être qualifiée de « prédictive ». Par ce terme, il faut comprendre que il n'y a pas, de certitude totale dans le processus d'extraction et de traduction des données. Il n'est pas possible de récupérer avec certitude le bon identifiant. L'objectif cet algorithme n'est donc pas de trouver la « bonne » réponse. Il est de construire une chaîne de caractère dont on prédit qu'elle apportera un résultat pertinent. De la même manière, la phase de préparation des données est un processus qui sélectionne et normalise certaines informations dont on considère – après un long processus de tests et d'essais – qu'elles

seront pertinentes dans l'obtention des bons résultats. Enfin, le premier rôle des tests est de quantifier les prédictions. Ils répondent à la question : étant donné les résultats obtenus lors des tests, quelle est la probabilité que la prochaine chaîne de caractères recherchée retourne un résultat pertinent ? Cette approche prédictive implique nécessairement un degré d'incertitude, et donc le développement d'algorithmes flexibles qui cherchent à minimiser le bruit.

Être conscient de la nature prédictive de ce processus et quantifier la qualité des algorithmes à l'aide de tests permet cependant de prendre de meilleures décisions techniques. La lecture distante et la détection de motifs supposent d'avancer « à l'aveugle », en s'appuyant sur sa connaissance de la structure du texte pour extraire les bonnes informations. Étant donné qu'il est impossible d'être totalement certain que les bonnes données ont été extraites, l'étape suivante – le lancement des requêtes sur l'API – doit être malléable et s'adapter aux données disponibles. C'est pourquoi le parti pris a été de concevoir un algorithme qui continue de faire des requêtes en modifiant les paramètres utilisés tant qu'un identifiant n'a pas été trouvé.

Chapitre 4

Un algorithme de détection de motifs pour préparer et structurer les données

Avant de chercher à récupérer un identifiant *Wikidata* via l'API, un algorithme se charge de traduire et de structurer les données : à partir d'un nom et de son éventuelle description, un dictionnaire qui contient les informations de manière structurée est construit. Cette étape était initialement censée être une simple extraction d'information : à partir du `tei:name` et du `tei:trait`, un ensemble d'informations étaient mises bout à bout afin de former une chaîne de caractères à rechercher sur l'API. Le processus s'est complexifié pour intégrer l'extraction, la traduction et la structuration de données présentes dans les catalogues. En construisant un dictionnaire à partir de texte, il est possible de savoir précisément quelles données sont disponibles pour lancer des requêtes ; plusieurs requêtes peuvent alors être lancées sur l'API avec différents paramètres, ce qui permet d'augmenter les probabilités d'obtenir un identifiant valide.

4.1 Présentation générale

4.1.1 Les formats d'entrée et de sortie

Le but de l'extraction de données permet de transformer la représentation `TEI` visible en 4.1 – représentée sous forme d'un `TSV` pour faciliter la lecture des données – en un dictionnaire visible en 4.2. Voici la significtion des différentes clés [1] du format de sortie :

— `fname` : cette clé permet d'accéder au prénom d'une personne. Les données contenues dans cette clé viennent du `tei:name`. `fname` est l'abréviation de *first name*.

1. Une clé de dictionnaire est l'élément à gauche des « : » ; la clé permet d'accéder à la valeur, visible à droite du « : » ; la clé permet donc de définir le sens de la valeur.

```xml
<item n="271" xml:id="CAT_000327_e271">
    <!-- ... -->
    <name type="author">Turenne (Henri de La Tour d'Auvergne vicomte
    ↪ de)</name>
    <trait>
        <p>illustre maréchal de France, né en 1611, tué en 1675.</p>
    </trait>
    <!-- ... -->
</item>
```

Code source 4.1 – L'entrée `XML-TEI` à partir de laquelle des données sont extraites

— `lanme` : cette clé permet d'accéder au nom de famille de quelqu'un. C'est cette information, extraite du `tei:name`, qui est centrale aux requêtes. `lname` abréviation de *last name*)

— `nobname_sts` : cette clé contient un nom de famille noble. Dans ces cas, un titre de noblesse est présent dans les entrées de catalogue (seuls les titres de noblesse les plus importants sont extraits du dictionnaire, ce qui n'est pas le cas ici). Les informations contenues ici proviennent du `tei:name`. Cette clé est l'abréviation de *nobility name_status*

— `status` : le statut d'une personne, soit son titre de noblesse (ici, le titre « vicomte » n'a pas été extrait car il est rarement présent sur *Wikidata*). Les informations contenues ici proviennent en général du `tei:name` et parfois du `tei:trait`.

— `dates` : les dates de naissance ou de mort d'une personne (seules ces dates sont conservées). Ces informations proviennent du `tei:trait`.

— `function` : la fonction d'une personne, soit, en général, son métier ou son occupation principale. Cette information provient du `tei:trait`.

— `rebuilt` : un booléen indiquant si un prénom a été reconstruit à partir d'initiales ou non.

Comme cela a été dit auparavant, le nom attribué à ces clés n'est pas systématiquement indicateur des valeurs qui y sont associées : si l'entrée de catalogue correspond à une personne, alors les clés définissent correctement les valeurs qui y sont associées. Si l'entrée de catalogue ne correspond pas à une personne, ces clés seront également utilisées. Ce qui est important, c'est la hiérarchie d'importance entre les différentes clés : `lname` est la clé centrale et contient presque toujours des informations, `fname` des données secondaires et `date` des dates. Les autres clés sont rarement utilisées si l'entrée de catalogue ne correspond pas à une personne.

```
{
    "fname": "henri ",
    "lname": "la tour d'auvergne",
    "nobname_sts": "Turenne ",
    "status": "",
    "dates": "1611 1675 ",
    "function": "marshal",
    "rebuilt": False
}
```

Code source 4.2 – La sortie `JSON` correspondante

4.1.2 Présentation de l'algorithme d'extraction d'informations

L'algorithme détaillé ci dessous est présenté sous forme graphique dans la figure 4.1. Cette étape peut être séparée en deux parties différentes : l'extraction d'informations nominatives du `tei:name` et la récupération de données biographiques du `tei:trait`.

L'extraction d'informations du `tei:name` est l'étape plus complexe. La difficulté tient au fait que cette balise peut contenir des informations variées et structurées de façon très différente. Cela demande d'identifier des motifs récurrents et de les repérer dans le texte à différents degrés et à différentes étapes. Dans un premier temps, les prénoms sont systématiquement extraits. Ils sont toujours détectés – même si l'élément ne contient pas le nom d'une personne : cette extraction permet justement d'identifier le type de données contenues dans le `tei:name`. Les prénoms sont repérés à l'aide de plusieurs expressions régulières qui permettent d'identifier des prénoms complets et abrégés, qu'ils soient composés ou non. Cette détection prend en compte les différents types d'abréviations possibles (un prénom composé peut être entièrement abrégé ; à l'inverse, seulement un des prénoms peut être abrégé) ; elle s'appuie sur les usages variables de caractères typographiques (séparer les prénoms avec des traits d'union ou non, par exemple). Dans le cas où un prénom serait abrégé, il est si possible reconstruit : des initiales sont remplacées par un nom complet ; ce processus est présenté plus en détail ci-dessous. Cela permet d'augmenter le taux de réussite dans l'alignement avec des identifiants *Wikidata*, mais vient avec plusieurs difficultés techniques, comme nous le verrons. Une fois ce nom extrait, le type d'information contenue dans le `tei:name` doit être identifié : en fonction du type d'information (géographique, historique, nominative...), différents traitements sont mis en place. Cette identification se fait par détection de motifs augmentée par l'usage de tables de conversion [2] et de listes contenant du vocabulaire spécifique. Listes et tables étant classées thématiquement, il est possible, par un processus éliminatoire, d'identifier avec certitude le type d'information contenue dans le `tei:name`. Le traitement du `tei:name` dépend grandement de cette détection : si cette balise contient des éléments

2. Pour un exemple de table de conversion, voir 1

géographiques ou historiques, l'extraction d'informations repose en grande partie sur les tables de conversion. S'il s'agit en revanche d'un nom de personne, il est alors nécessaire d'identifier les différents types de données nominatives (prénom, nom de famille, nom de famille noble...) pour bien structurer les données. En effet, c'est de cette structure que dépend la bonne construction du dictionnaire, et donc la constitution de requêtes adaptées à l'API. Ce processus d'extraction des données s'appuie majoritairement sur une détection de motifs. Le motif déterminant est la présence ou non de parenthèses dans le nom. Comme nous le verrons, si un nom contient des parenthèses, les informations sont bien plus structurées que s'il n'en contient pas. Les informations peuvent alors être extraites avec une bien plus grande granularité.

Une fois les informations nominatives extraites du `tei:name`, il reste à extraire les données biographiques pertinentes du `tei:trait`. Cette étape, plus simple que la précédente, vaut principalement pour les entrées où c'est l'auteur.ice d'un document qui est mentionné.e dans le `tei:name`. Les seules informations extraites concernent les dates de vie et de mort des personnes, ainsi que leur métier. Quelques difficultés techniques subsistent cependant : il faut notamment distinguer une date de naissance/décès d'une autre date, afin de diminuer le bruit ; de plus, il faut réussir à extraire de façon automatique l'occupation principale d'une personne, lorsque plusieurs personnes sont mentionnées (« militaire » et « auteur », par exemple). La résolution de ces deux problèmes repose sur une bonne connaissance du corpus de catalogues et de la structure des `tei:trait`.

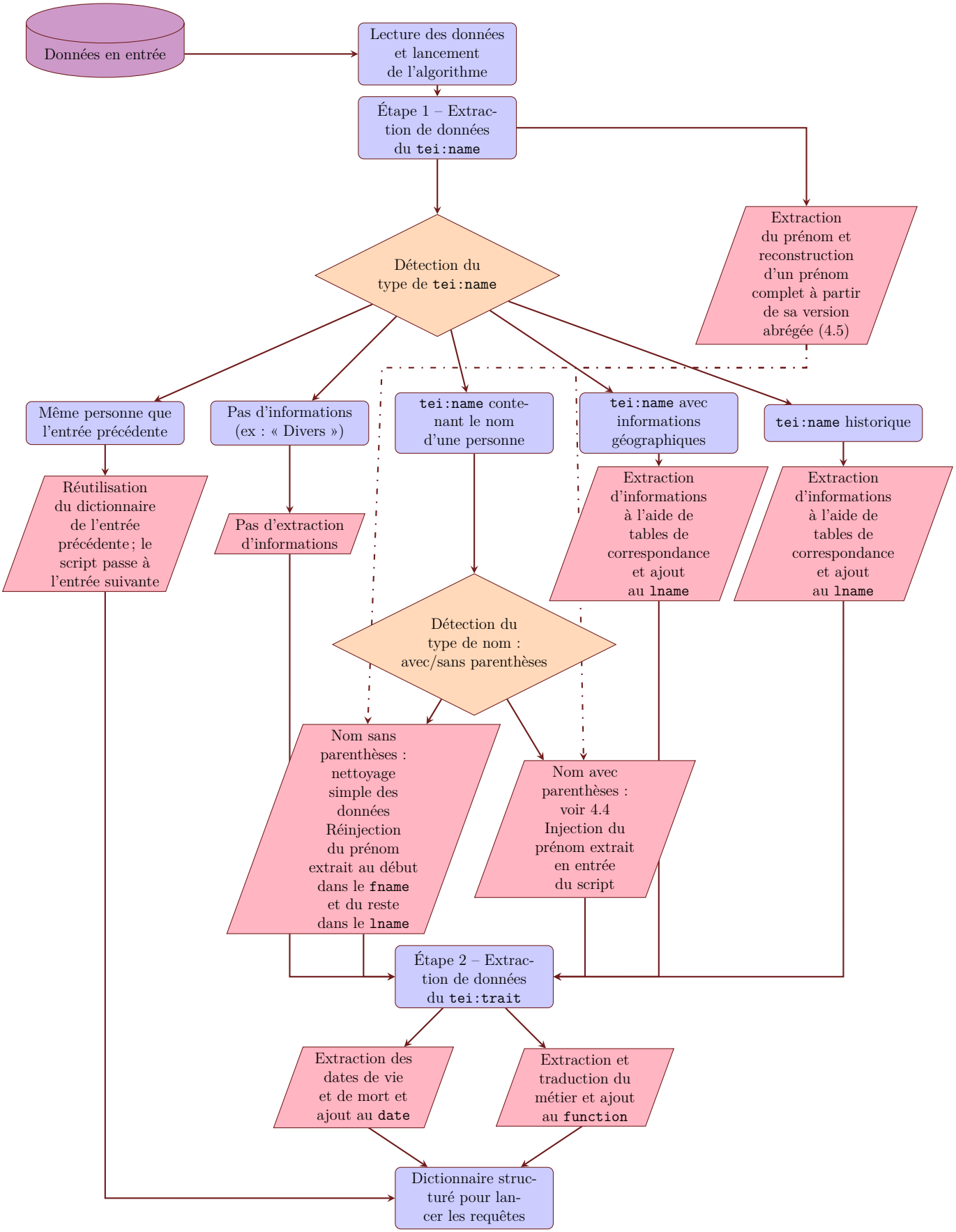


FIGURE 4.1 – Processus d'extraction d'informations du `tei:name` et `tei:item`

4.2 Identifier le type de nom

Les éléments `tei:name` contiennent le titre donné à l'item vendu. Si c'est souvent le nom de l'auteur.ice du document, ce n'est pas toujours le cas. L'extraction d'éléments du `tei:name` dépend, comme cela a été dit, de l'identification du « type » de nom. La décision a été prise de classer tous les `tei:name` en cinq catégories.

4.2.1 Les différents types de noms

Les noms génériques

Ces éléments ne contiennent pas d'informations précises. Les entités *Wikidata* étant spécifiques plutôt que génériques, il n'est pas certain que les entrées puissent être alignées avec *Wikidata* ; si des entités *Wikidata* correspondent à ces éléments, les informations qu'elles contiennent seront probablement trop génériques pour être utilisables dans un contexte économétrique. Lorsque cela est possible, l'alignement est quand même fait ; c'est le cas par exemple pour les `tei:name` contenant la mention de chartes. Si les informations sont trop génériques (comme dans l'exemple ci-dessous), un dictionnaire vide est retourné et l'alignement avec *Wikidata* n'a pas lieu. Dans cette catégorie se trouve par exemple :

— `<name type="author">DIVERS</name>`

Les noms de type « le/la même »

Lorsqu'il y a cette valeur dans le `tei:name`, l'auteur.ice est la même personne que l'auteur.ice de l'entrée de catalogue précédente. Dans ce cas, le dictionnaire de cette entrée est réutilisé. Par exemple :

— `<name type="author">Le même</name>`.

Les noms géographiques

Ces entrées sont détectées à l'aide de tables de conversion et de listes. Celles-ci sont classées thématiquement : liste d'anciennes colonies françaises (annexe 4), de départements français du XIX$^{\text{ème}}$ s.(annexe 2), d'anciennes provinces françaises (annexe 5) et de pays (annexe 3). Dans ce cas, un alignement avec une entité *Wikidata* est possible ; cependant, il n'est pas toujours envisageable de retrouver l'entité précise. En effet, un `tei:name` peut contenir une mention d'une donnée géographique, mais également d'autres détails. C'est par exemple le cas dans le quatrième exemple ci-dessous. Il faut alors aligner le `tei:name` avec son équivalent générique sur *Wikidata* (l'exemple ci-dessous, par exemple, a été aligné uniquement avec l'entité « Paris »). Dans cette catégorie se trouvent :

— `<name type="other">AISNE (département de 1')</name>`

— `<name type="author">Bourbonnais. </name>`

— `<name type="author">Paris : Musée royal du Louvre</name>`

— `<name type="author">Garde nationale parisienne en 1792 (brevet de volontaire de la)</name>`

Les noms correspondant à des évènements historiques

Là encore, une table de conversion est utilisée (6). Ici, une difficulté apparaît cependant : du fait de la varité des évènements historiques mentionnés dans les entrées de catalogue, il n'est pas possible d'enregistrer l'ensemble des évènements dans des tables afin de permettre une détection de tous les évènements. Les `tei:name` ont donc été analysés pour extraire les évènements les plus importants. Ensuite, comme pour les termes géographiques, il n'est pas possible de donner aux tables de conversion une granularité suffisamment fine pour contenir toutes les données possibles. Des alignements partiels ont donc été faits : le premier exemple ci-dessous a été aligné avec l'entité *Wikidata* « Révolution française ».

— `<name xmlns="http://www.tei-c.org/ns/1.0" type="other">THÉATRE RÉVOLUTIONNAIRE</name>`

— `<name type="author">COMMUNE DE 1871.</name>`

— `<name type="other">Siège de La Rochelle en 1628</name>`

Les noms de personnes

Ceux-ci ne sont pas simples à traiter : ils peuvent contenir de nombreuses informations : deux noms de famille (usuels et nobles), titres de noblesse, plusieurs prénoms. Ils ont également une structure variée, comme cela apparaît dans les exemples ci-dessous : les noms peuvent être écrits en utilisant des parenthèses ou non ; un prénom peut être écrit en entier, comme dans premier exemple, entièrement abrégé (comme dans l'exemple 3) ou encore partiellement abrégé, ce qui est le cas du dernier exemple. Ces différences, qui ne posent pas de problème à un regard humain, sont autant de problèmes techniques. En effet, la détection de motifs fonctionne uniquement sur des caractéristiques formelles ou structurelles du texte. Il faut donc, en s'appuyant sur la structure des documents, réussir à distinguer un prénom d'un nom de famille, un nom de famille d'un autre, ou encore un nom abrégé d'un nom complet.

— `<name type="author">Humboldt (le baron Alexandre de)</name>`

— `<name type="author">Taccani Tasca (madame la comtesse)</name>`

— `<name type="author">LEGOUVÉ (G. M. J. B.)</name>`

— `<name type="author">LOUIS XVIII</name>`

— `<name type="author">Duras (Emm.-F. de Durfort, duc de)</name>`

4.2.2 Une méthodologie pour distinguer les différents types de noms par élimination

Si l'on pose l'extraction d'informations nominatives sur une personne comme étant l'objectif principal, une difficulté apparaît vite : qu'est-ce qui distingue un nom de personne des autres types de noms ? Les éléments `tei:name` sont voués à contenir des noms propres ; chercher à distinguer les noms propres des noms communs n'a donc pas d'intérêt. Cela est d'autant plus vrai que la graphie varie d'un catalogue à l'autre : dans certains, les majuscules sont signifiantes et pourraient permettre de distinguer noms communs et noms propres, tandis que dans d'autres, l'intégralité du `tei:name` est en majuscule. Une détection de motifs à partir de simples critères formels (du type : « Un nom de personne est un ou plusieurs mots commençant par des majuscules ») n'est pas opérante pour ce corpus. À cette étape, il n'est pas non plus possible de s'appuyer sur l'extraction de prénoms. Celle-ci se base, comme on le verra, sur de la détection de motifs ; là encore, ce qui est identifié comme un prénom peut tout aussi bien être n'importe quel autre nom propre, et il n'existe à ce stade aucune possibilité pour distinguer un nom propre d'un autre. Il est donc impossible de définir un nom positivement, puisqu'il n'existe aucun critère définitoire pour un nom propre. C'est à ce stade qu'a été décidée l'utilisation de tables de conversion et de listes contenant du vocabulaire thématique : en identifiants des références récurrentes à des évènements ou des lieux dans les `tei:name`, il devient possible de définir les noms de personne négativement. En définitive, un nom de personne, c'est donc ce qui n'est pas autre chose et la détection du type de nom fonctionne donc de manière éliminatoire.

L'algorithme mène donc une série de tests, cherchant à détecter si un `tei:name` rentre dans telle ou telle catégorie. Pour des raisons techniques, une série de tests éliminatoires doit aller du cas le plus particulier au cas le plus générique afin d'éviter les faux positifs : une fois qu'un élément a été détecté comme appartenant à une catégorie, il ne peut plus être réassigné à une autre. C'est pourquoi l'algorithme commence par chercher à classer les éléments dans les catégories où le taux d'erreur est le plus faible, pour ensuite finir par la catégorie la plus générique : celle des noms de personne. : c'est dans ces catégories que le taux d'erreur est le plus faible.

Le script commence donc par chercher à classer un `tei:name` dans les catégories où les informations sont écrites plus ou moins toujours de la même manière. Il cherche d'abord à identifier si le `tei:name` correspond à « Le même » ou « La même ». Dans ce cas, le `tei:name` est le même que celui de l'entrée précédente et c'est le dictionnaire produit pour celle-ci qui est réutilisé. Ensuite, les entrées génériques sont détectées. Comme elles contiennent toujours des informations écrites de la même manière (« Documents divers »), le taux d'erreur est là encore très faible. Ces entrées génériques ne contiennent pas d'informations spécifiques ; si un `tei:name` appartient à cette catégorie, alors l'algorithme

n'extrait pas d'informations.

Ensuite, l'algorithme cherche à classifier un nom en différentes catégories à l'aide de tables de comparaison et de listes contenant du vocabulaire spécifique. Ces tables et listes sont classées en deux catégories (données historiques et géographiques) afin de définir des traitements spécifiques ; l'algorithme cherche d'abord à identifier des entrées géographiques, puis historiques. En effet, un bien plus grand nombre de tables contenant des données géographiques existe [3], ce qui augmente les possibilités d'un classement correct. Enfin, le script cherche à identifier des informations historiques dans un `tei:name` (6). Si une donnée géographique ou historique est repérée, alors équivalents normalisés de cette donnée sont ajoutés au dictionnaire grâce à l'usage de tables de conversion.

Par processus d'élimination, si aucune de ces informations n'a été détectée, alors il n'est plus possible de classer le `tei:name` dans aucune autre catégorie. Il est alors considéré que le contenu du `tei:name` est le nom d'une personne. L'extraction d'informations se fait ici plus complexe [4], comme nous le verrons : le script traite le nom différemment en fonction de sa structure (présence ou non de parenthèses dans le nom), puis extrait d'éventuels titres de noblesse, noms de famille noble et nom de famille usuel. Enfin, il extrait les prénoms et cherche à reconstruire un prénom complet à partir de son abréviation. Le processus étant éliminatoire, il n'y a bien sûr aucune certitude que le contenu du `tei:name` soit bel et bien le nom d'une personne ; il n'est cependant plus possible de mieux catégoriser les éléments. Cela n'est pas non plus extrêmement important, puisque cet algorithme de classification a un rôle fonctionnel et n'impacte pas la manière dont les identifiants sont récupérés depuis l'API *Wikidata*. Il permet principalement d'adopter un fonctionnement modulaire en cherchant à détecter des motifs spécifiques dans les `tei:name` en fonction de la catégorie à laquelle ils appartiennent. La détection de motifs étant « aveugle », le traitement qui est fait ici peut être mis à profit de différents types de données : les mêmes motifs peuvent être recherchés et extraits de différents types d'entrées.

À l'issue de cette phase de classification, il est possible de mieux connaître les types de `tei:name` et leur répartition dans le corpus. Les noms de personnes sont très majoritaires : ils représentent 80634 entrées sur un total de 82913, soit 97,25% du corpus. C'est donc pour ce type de données que le processus d'extraction a été pensé, ce qui permet d'extraire des informations avec une granularité plus fine qu'avec le reste du corpus. Viennent ensuite les noms de lieux (1406 entrées) et les éléments divers (550). Pour finir se trouvent les évènements historiques (232 entrées) et les éléments vides, soit 92 entrées [5]. Bien que,

3. Nom d'anciennes provinces françaises (5), de départements du XIX$^{\text{ème}}$ s. (2), d'anciennes colonies (4) et de pays (3)

4. Pour une représentation graphique de cette étape, voir 4.4

5. Ces chiffres proviennent d'un calcul réalisé à partir du script développé pour identifier le type d'entrée. Ils ont été produits afin de réaliser une représentation graphique du type d'entrée, visible en annexes (3)

comme cela a été dit, la classification des `tei:name` en différents types soit purement fonctionnelle, ces chiffres permettent d'avoir une meilleure connaissance du corpus, et mieux comprendre à partir de quel type de données se fait l'alignement avec *Wikidata*. Cette classification montre également d'intérêt de la détection de motifs augmentée de tables de correspondances : bien que ce soit une méthode « légère », elle peut efficacement, sur un corpus semi-structuré, être à la base d'une extraction précise de données.

4.3 Le traitement des noms de personnes

Comme cela a déjà été dit, le processus d'extraction d'informations du `tei:name` est relativement simple pour les entrées qui ne contiennent pas de noms de personnes : il s'agit principalement de remplacer des informations non-normalisées par leur équivalent normalisé, à l'aide d'un dictionnaire. Pour les noms de personne cependant, le processus est plus complexe : il demande d'identifier titres de noblesse et différents types de noms ; de plus, de nombreux prénoms sont abrégés dans les catalogues, afin d'économiser de l'espace. Pour augmenter le taux de réussite de l'alignement avec *Wikidata*, une méthode a été développée pour reconstruire un nom complet à partir de son abréviation. Après avoir présenté les différentes étapes de l'extraction d'informations nominatives, cette partie détaillera ce processus de reconstruction du prénom. Dans ces deux étapes, la nature semi-structurée des documents est centrale, puisqu'elle permet d'identifier avec une grande précision les différentes informations.

4.3.1 Trouver des solutions adaptées à différents types de noms

Pour extraire et structurer des informations d'un `tei:name`, il est nécessaire d'identifier ce qui distingue un prénom d'un nom de famille ou d'un titre de noblesse. Une première difficulté apparaît vite : formellement, il n'y a pas nécessairement de différence entre ces informations. Dans l'exemple 4.1, le `tei:name` correspond à :

```
1 <name type="author">Turenne (Henri de la Tour d'Auvergne vicomte
  ↪ de)</name>
```

Code source 4.3 – Le `tei:name` de l'exemple 4.1

Les informations à extraire de cet élément sont visibles dans la figure 4.2. Si chaque mot est pris individuellement, il est difficile de les distinguer l'un de l'autre. Le nom de famille noble, le prénom et le nom de famille usuels débutent tous par une majuscule. Il est ici impossible de trouver un motif distinctif pour un nom de famille ou pour un prénom. Cela est d'autant plus vrai qu'un nom peut être composé ; ici, le nom de famille usuel est composé de deux mots débutant par une majuscule et séparés par un « d' ». Étant donné

la taille du corpus, il n'est pas non plus possible d'utiliser des tables de conversions pour identifier ce qui est un nom ou un prénom, comme cela a été fait pour les lieux et les évènements géographiques. La seule information qui peut être traitée à l'aide d'une table de conversion est le titre de noblesse, puisque ces titres existent en nombre limité. Par conséquent, aucun élément à l'intérieur des termes à relever ne permet de les identifier. Là où ils peuvent cependant être distingués, c'est au niveau de leur place relative au sein de la phrase. Pour reprendre l'exemple 4.2, le nom de famille noble se retrouve hors de la parenthèse, au tout début de la phrase ; les autres informations sont dans la parenthèse, où le prénom est suivi du nom de famille et du titre. S'il n'est pas possible d'identifier la valeur d'un nom à partir de ses caractéristiques propres, il est donc possible de la déduire de sa position dans la phrase. C'est ici qu'une bonne compréhension de la structure des `tei:name` devient essentielle.

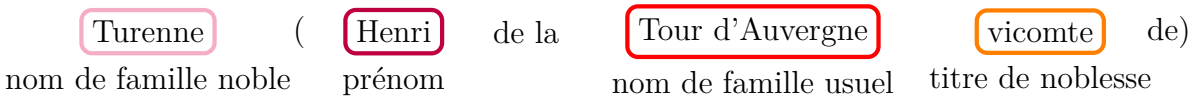


FIGURE 4.2 – Les différentes parties du `tei:name`

Il est donc nécessaire d'identifier les différentes structures possibles pour un `tei:name` contenant un nom de personne. Celles-ci sont visibles sous forme graphique dans la structure 4.3. Pour les besoins de l'algorithme, deux structures principales ont été retenues : les `tei:name` contenant des parenthèses et eux qui n'en contiennent pas [6]. Comme nous le verrons, d'autres subdivisions existent ensuite.

Les `tei:name` contenant des parenthèses ont une structure bien plus claire que ceux n'en contenant pas ; il est alors tout à fait possible d'identifier le rôle joué par les différents éléments de cette balise. L'organisation entre les différents types de noms est la même dans les exemples 4.4 et 4.5, qui contiennent tous les deux des parenthèses. Le nom le plus important, sous lequel une personne est connue au XIX$^{\text{ème}}$ s., est contenu entre parenthèses et au début de l'entrée, parfois en majuscules. Le ou les noms suivants sont, eux, à l'intérieur de parenthèses. Le premier nom au sein des parenthèses est toujours le prénom ; ensuite viennent des informations complémentaires. Ces deux exemples, cependant, forment deux sous-catégories dans le groupe des noms sans parenthèses :

— Les exemples visibles en 4.4 représentent uniquement des noms de personnes nobles. Dans ce cas, le nom hors parenthèses correspond au nom de famille noble ; le nom de famille usuel d'une personne est contenu à l'intérieur des parenthèses. Ce détail est important, parce que les personnes sont souvent référencées sur *Wikidata* par le nom de famille usuel. Lancer une recherche sur l'API avec les noms de famille usuel et noble à la fois ne retournera pas toujours de réponse, alors que ne rechercher qu'un des deux noms peut permettre d'obtenir le bon résultat. C'est pourquoi, dans

6. Comme cela étant déjà visible dans la figure 4.1.

ces cas, les deux noms sont distingués : le nom de famille usuel est associé avec la clé `lname` du dictionnaire, et le nom noble lié au `nobname_sts`.

— Les personnes dans le second exemple (4.5) ne sont pas nobles, ce qui simplifie beaucoup l'extraction d'information. Il suffit de stocker la partie en dehors des parenthèses dans le `lname` ; le prénom entre parenthèses, identifié grâce à l'algorithme d'extraction des prénoms, est stocké dans le `fname`. Parfois, d'autres informations sont contenues dans les parenthèses ; si elles sont signifiantes (comme dans le cas d'Alexandre Dumas père, où il est important de faire la distinction d'avec son fils), elles sont extraites. Sinon, ces informations supplémentaires sont abandonnées.

```
<name type="author">Turenne (Henri de la Tour d'Auvergne vicomte
↪ de)</name>
<name type="author">HUMBOLDT (Alexandre baron de)</name>
<name type="author">Tascher (Pierre-Jean-Alexandre-Jacquemin comte
↪ Imbert de)<name>
```

Code source 4.4 – Trois exemples de `tei:name` avec titres de noblesse

```
<name type=author>Viardot (Pauline)</name>
<name type=author>Verdi (Giuseppe)</name>
<name type=author>Sobieski (Thérèse-Cunégonde)</name>
<name type="author">CAUCHY (le bon Alex.)</name>
<name type="author">DUMAS (Alex. père)</name>
```

Code source 4.5 – Cinq exemples de `tei:name` sans titres de noblesse

Les `tei:name` ne contenant pas de parenthèses (exemple 4.6) ne peuvent être traités avec la même granularité, puisqu'ils ne présentent pas de caractères récurrents pour faire la différence entre leurs différentes parties. Il est donc impossible de s'appuyer de façon récurrente sur la position des différents noms dans la phrase pour déduire leur signification. Cependant, il faut également remarquer que les éléments ne contenant pas de parenthèses contiennent en général bien moins d'informations que ceux qui en contiennent. Chercher à tout prix à identifier une structure à ces entrées afin de pouvoir avoir un dictionnaire complet à partir duquel lancer plusieurs requêtes n'a pas non plus nécessairement d'intérêt.

```
<name type="author">Sophie</name>
<name type="author">Henri VI</name>
<name type="author">DUPUIS</name>
```

Code source 4.6 – Trois exemples de `tei:name` sans parenthèses

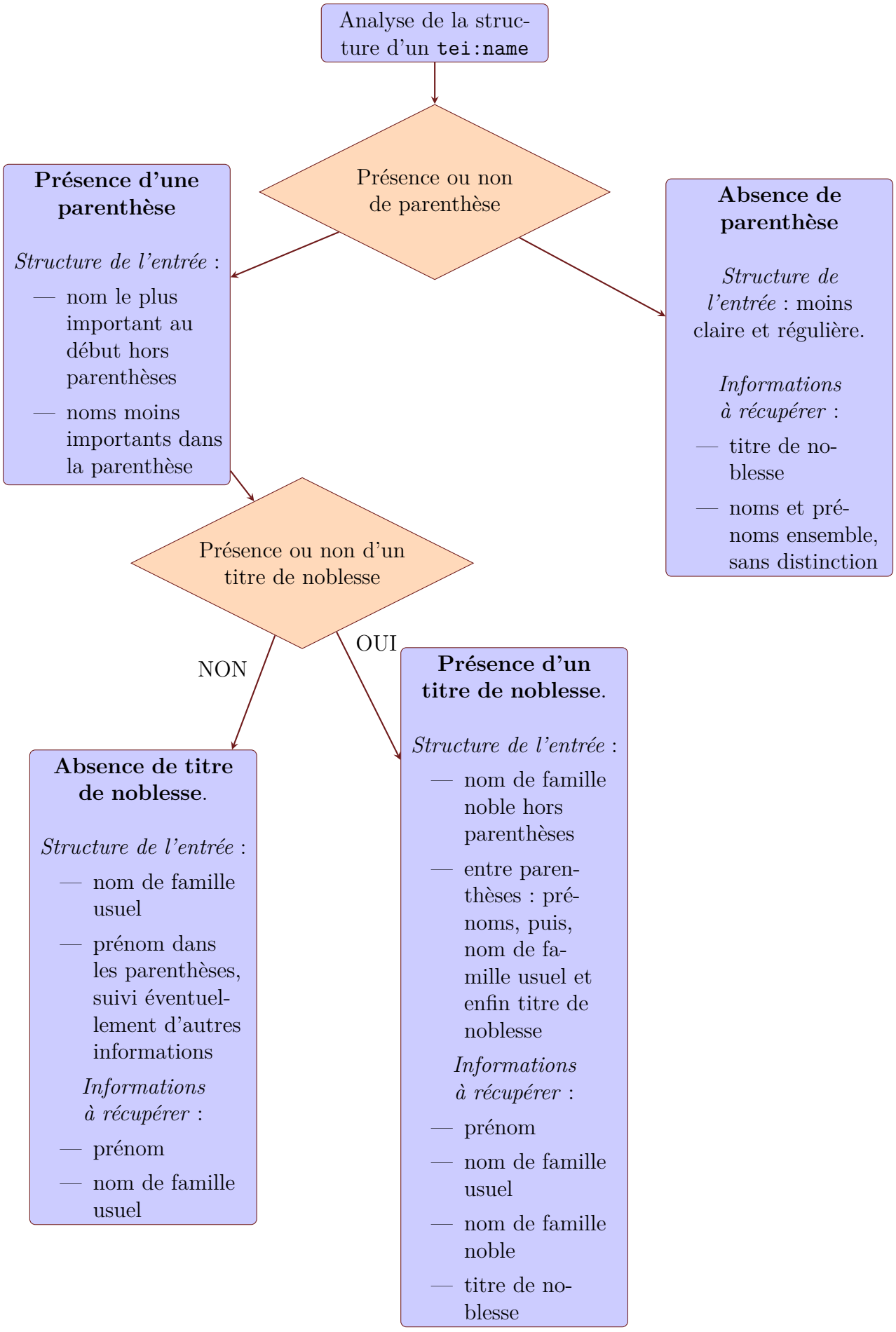


FIGURE 4.3 – Différentes structures possibles pour un `tei:name`

4.3.2 Identifier et extraire les informations nominatives

C'est sur les différents critères présentés dans la partie précédente et résumés sous forme graphique dans la figure 4.3, que se base l'extraction d'informations nominatives du `tei:name`.

Dans un premier temps, l'algorithme cherche à extraire des informations d'un `tei:name` contenant des parenthèses (étape représentée dans la figure 4.4). Il y a alors quatre données à identifier et à extraire : le nom de famille noble, le nom de famille usuel, le prénom et le titre de noblesse. L'algorithme commence par extraire des informations du texte contenu entre parenthèses avant de passer au texte hors parenthèses : il identifie un titre de noblesse avant d'extraire un prénom et éventuellement un nom de famille ; pour finir, le nom hors parenthèses est extrait.

L'extraction du titre de noblesse est l'étape la plus simple, puisqu'elle repose sur une table de conversion (7) : l'algorithme cherche à identifier un titre de noblesse parmi les clés de ce dictionnaire. Si des titres sont trouvés, alors leur version normalisée figurant en valeur de ce dictionnaire est ajoutée aux données extraites. Il est à noter que, comme tous les titres de noblesse les plus importants permettent d'obtenir des résultats sur *Wikidata*, aucune valeur n'est associée aux titres les moins importants ; dans ce cas, aucun titre n'est extrait, puisqu'il risque d'empêcher l'obtention d'un résultat. Cependant, même des titres de noblesse n'ayant pas d'équivalent, et qui ne seront donc pas extraits, figurent sur cette table de conversion : la table permet d'identifier la présence d'un titre de noblesse, ce qui est nécessaire pour extraire les autres informations.

Si un titre de noblesse a été trouvé, un processus éliminatoire est engagé afin d'identifier le prénom et le nom de famille usuels contenus entre parenthèses. Comme cela apparaît dans les exemples plus haut, le texte entre parenthèses peut contenir du bruit : des mots comme « du, le »... qui n'aident pas à obtenir de résultats sur *Wikidata*, ou encore des noms communs et des attributs des personnes. Il faut détecter tous les noms propres, situer leur rôle et les extraire. Il peut être tentant d'extraire seulement les noms propres à l'aide d'expressions régulières – c'est à dire, d'identifier tous les mots commençant par une majuscule – et d'abandonner le reste. Cependant, certains noms communs peuvent être écrits avec une majuscule ; en suivant cette méthode, il y aurait donc un risque d'inclure du bruit dans les données utilisées pour lancer les requêtes. L'algorithme lance donc une série de simplifications du contenu entre parenthèses : il commence d'abord par supprimer les données déjà extraites : le prénom ou son abréviation identifiées et le titre de noblesse ; ensuite, il supprime l'ensemble des termes « auxilliaires » (« le, la, dit... »), qu'ils aient des capitales ou non ; enfin, tous les noms communs restants sont supprimés. Il ne reste alors du texte entre parenthèses qu'un ou plusieurs noms commençant par une majuscule et considérés comme des noms propres ; quelle valeur donner à ce texte ? Étant donné que le prénom a déjà été identifié, de même que le titre de noblesse, le texte entre parenthèses

ne peut être qu'un nom de famille ; on considère alors que le texte entre parenthèses est un de nom de famille usuel ; le texte hors parenthèses est alors le nom de famille noble.

Si un `tei:name` contient des parenthèses mais pas de titre de noblesse, le processus est analogue à celui décrit ci-dessus – mais plus simple, puisque le texte entre parenthèses a tendance à être bien plus complexe pour les personnes nobles. Le bruit est successivement détecté et supprimé à l'aide d'expressions régulières ; le prénom déjà extrait est supprimé. À ce stade, une deuxième extraction de prénoms a lieu : les prénoms étant parfois complexes et écrits d'une manière variable (avec ou sans parenthèses...), il arrive que ceux-ci ne soient pas entièrement extraits. Cette seconde extraction pose une légère difficulté, puisque l'on dispose alors de deux noms propres ayant la valeur d'un prénom. Il faut déterminer la relation entre ces deux « prénoms » : les mettre simplement bout-à-bout risquerait de créer des noms et prénoms dans le désordre ; dans ces cas, le moteur de recherche de *Wikidata* ne retournerait pas de résultat. Par conséquent, c'est en fonction de la position relative des deux « prénoms » que le prénom final est reconstruit : le prénom est reconstruit en fonction de la position relative des deux prénoms qui ont été successivement extraits. Pour finir, c'est le texte hors parenthèses qui est extrait, soit le nom de famille. En l'absence de titre de noblesse, il n'est pas nécessaire de faire la différence entre différents noms de famille ; le nom hors parenthèses ne peut être qu'un nom de famille usuel.

Si un nom ne contient pas de parenthèses, le processus d'extraction est à la fois plus simple et moins qualitatif : il est tout simplement impossible de classer les différentes données extraites par détection de motifs. Dans ce cas là, le bruit supprimé du `tei:name` ; l'objectif est de pouvoir lancer des recherches sur *Wikidata* avec les données les plus propres possibles, afin de maximiser les possibilités d'obtenir un résultat correct.

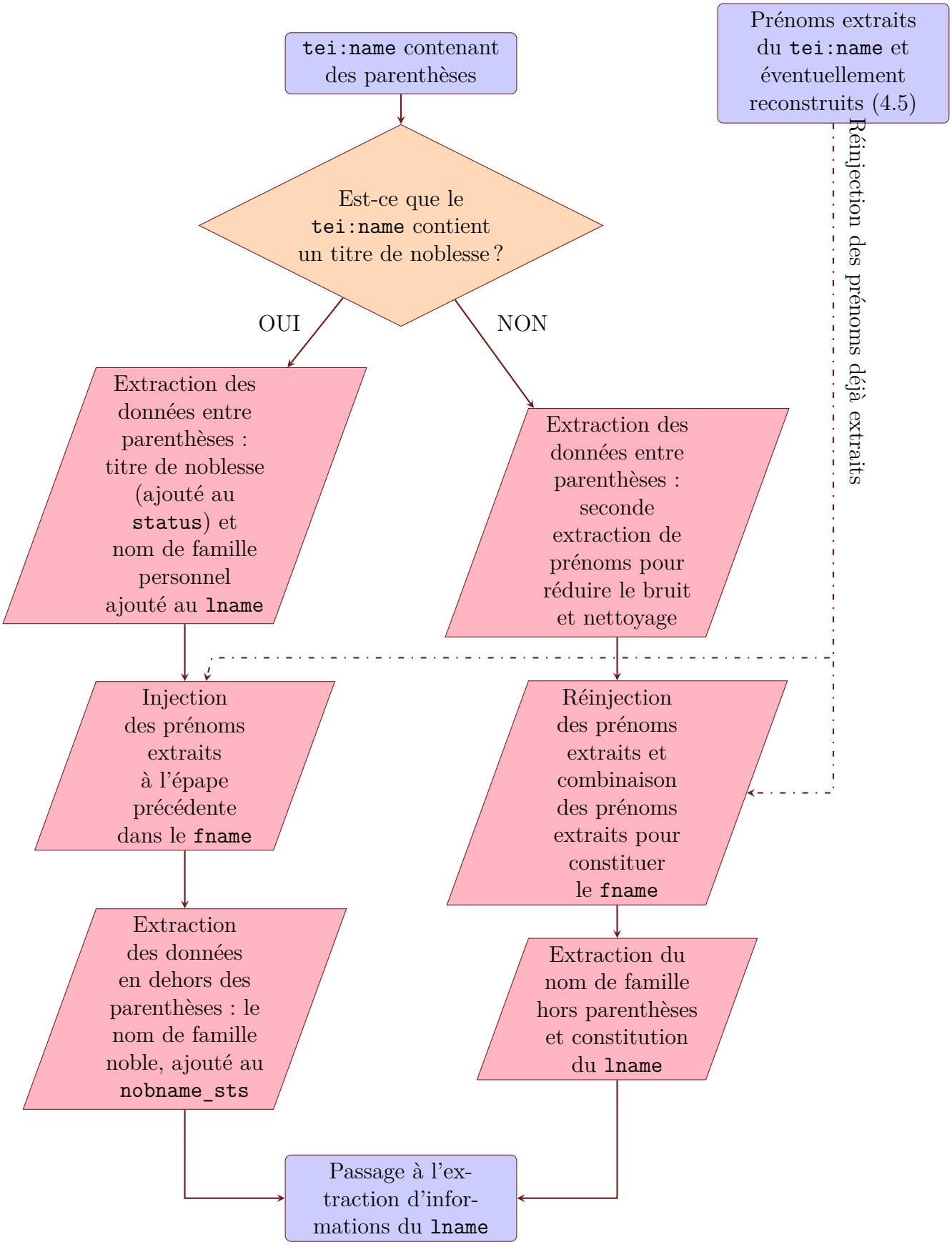


FIGURE 4.4 – Extraction d'informations d'un `tei:name` contenant des parenthèses

4.3.3 Reconstruire un prénom complet à partir de son abréviation

Cette phase essentielle vise à corriger un problème dans les données : le nom d'une personne est souvent abrégé. Cela pose un problème pour l'utilisation ultérieure de l'API : le prénom est une information essentielle pour identifier l'alignement avec la bonne entité sur *Wikidata*. Sans ce prénom, le risque est grand que le premier résultat obtenu à l'aide du moteur de recherche de *Wikidata* ne soit pas celui de la personne décrite dans le `tei:name`, mais celui d'un.e autre membre de la même famille. De plus, si une requête est lancée avec un prénom abrégé (les initiales d'une personne, par exemple), elle ne retourne pas de résultat [7]. Le choix a donc été fait, pour chaque entrée, d'essayer de reconstruire un prénom complet à partir de son abréviation. Comme cela a été montré plus haut, l'extraction du prénom est également essentielle pour identifier les autres informations du `tei:name` par élimination. La reconstruction des noms a lieu en trois étapes :

— D'abord, un prénom, abrégé ou complet, est extrait.

— Ensuite, le type de prénom est identifié. Pour cet algorithme, trois types de prénoms existent : les prénoms complets (composés ou non), les prénoms abrégés non composés et les prénoms abrégés composés. Le choix de faire une distinction entre les deux catégories s'explique par la différence dans le processus de reconstruction.

— Enfin, l'algorithme cherche à reconstruire le prénom à l'aide de tables de conversions.

La première question qui se pose est : qu'est-ce qui constitue un prénom ? Celui-ci correspond à un motif (un ou plusieurs noms propres), mais il est identifié par sa position dans la phase : le prénom est la première information contenue entre parenthèses – cela vaut autant pour les noms de personnes nobles que pour les autres. Pour être certain de cibler uniquement le prénom, cette étape n'a donc lieu que dans les `tei:name` contenant des parenthèses. Bien qu'un motif « prénom » puisse être identifié et que la position du prénom soit fixe, il est en fait assez difficile de modéliser ce motif de façon précise. Il faut définir un ou plusieurs motifs qui correspondent à tous les cas de figures, à toutes les formes que peuvent prendre un prénom. Voici quelques exemples pour mieux cibler cette difficulté (les prénoms sont en gras) :

— « Sobieski (**Thérèse-Cunégonde**) » : un cas classique : prénom composé où les deux noms sont complets et séparés par des tirets.

— « Zimmermann (**P.-J.-G.**) » : une autre forme simple : un nom composé où seules restent les initiales, écrites en majuscules, terminées par des points et séparées par des tirets.

— « DUBOIS-FONTANELLE (**J. Gaspard**) » : une forme un peu plus complexe : le nom est partiellement abrégé, avec une initiale et un nom complet ; cependant, il

7. Par exemple, voici une recherche sur Jean-Jacques Ampère. Si l'on recherche « J J Ampère », les résultats obtenus sont incorrects. En recherchant « Jean Jacques Ampère », le résultat obtenu est le bon.

n'y a pas de tiret qui sépare les deux prénoms, et il faut donc que les deux parties soient détectées toutes les deux dans le même prénom (sans quoi, il existe un risque de se retrouver avec des prénoms partiels).

— « DESCHAMPS (**Jh Fr. L.**) » : pour finir, une forme qui mélange les complexités : trois prénoms abrégés, mais qui ne contiennent pas qu'une initiale ; il ne sont pas tous terminés par des points et ne sont pas séparés par des tirets.

Cette variété tient autant du bruit dans les données que de différentes formes de notation dans le corpus. Elle complexifie en tout cas la définition, d'un point de vue formel, de ce qui constitue un prénom ; il est notamment difficile d'extraire l'intégralité d'un prénom composé, puisqu'il faut faire en sorte que les différents noms ou initiales soient reconnus comme faisant partie du même nom, qu'ils soient séparés par des tirets ou non. Dans l'abstrait.

— Un nom composé abrégé est une série de majuscules suivies d'autres lettres et/ou de points ; il doit contenir au moins un point et un tiret (un motif plus précis risque de ne pas identifier tous les cas de figure ; un motif plus généraliste risque d'inclure trop de bruit).

— Un nom simple abrégé correspond à une majuscule, suivie ou non d'autres lettres et terminée par un point.

— Un nom complet est une série de majuscules suivies de minuscules, séparées ou non par des tirets.

Identifier ces motifs « dans l'abstrait » est une étape importante, qui permet de mettre en place une implémentation technique. Cependant, il est impossible de construire un motif unique correspondant à tous les cas de figures. La décision a donc été d'accepter cette variété et ce bruit et de créer une série de motifs sous la forme d'expressions régulières, qui permettent de cibler différents cas de figure. [8]. À partir d'une connaissance du corpus et de sa structure, il est donc possible d'identifier tous les prénoms dans le texte, et d'automatiser cette identification par détection de motifs.

Une fois les prénoms identifiés, il s'agit de reconstruire ceux qui sont abrégés. Comme pour le reste du processus d'extraction d'informations, cette étape s'appuie fortement sur l'usage de tables de conversion. Le principe général est simple : des tables de conversion sous la forme de dictionnaires contiennent en clés des formes abrégées de noms, et en valeurs les formes complètes. Si un nom extrait se retrouve en clé de ces tables, alors il est remplacé par sa version complète. Malheureusement, l'implémentation technique n'est pas aussi facile, comme le montre la figure 4.5. En regardant les tables de conversions utilisées (8, 9), il apparaît que celles-ci sont assez courtes, et ne contiennent pas une

8. Les trois fonctions permettant l'extraction de prénoms sont visibles en annexes ; 10 permet d'identifier les noms composés abrégés, 11 identifie des noms abrégés non composés et 12 se consacre aux noms complets

grande variété de noms. Le problème avec l'usage de tables de conversion, c'est d'abord qu'elles doivent être faites à la main, et qu'il n'est pas possible de prendre toutes les abréviations possibles en compte ; mais le véritable problème est qu'une seule abréviation peut faire référence à plusieurs noms. C'est malheureusement le cas des abréviations les plus banales : « J. » peut à la fois signifier « Jean », « Jeanne », « Joséphine », « Josiane »... Le choix a donc été fait de ne retenir que les abréviations « évidentes » pour éviter d'introduire du bruit supplémentaire. De plus, quand un nom est modifié par l'algorithme de reconstruction, cela est indiqué à l'aide de la clé `rebuilt` du dictionnaire utilisé pour lancer des requêtes (4.2). Du fait de cette relative pauvreté des données, l'approche est ici totalement prédictive : il est impossible d'être certain d'obtenir le bon nom complet à partir de son abréviation ; il est seulement possible de prédire que le prénom reconstruit sera conforme au vrai prénom (tel qu'il est écrit sur *Wikidata*). Par son fonctionnement modulaire, qui s'adapte aux différents cas de figure, l'algorithme de reconstitution des prénoms cherche à maximiser cette certitude.

L'algorithme de reconstruction des prénoms a été conçu pour chercher à palier à cette relative pauvreté des données de conversion, surtout dans le cas des noms composés abrégés. Ceux-ci posent en effet un problème au delà de leur simple extraction : presque toutes les combinaisons de prénoms sont possibles. L'algorithme cherche donc à reconstruire un prénom composé en plusieurs étapes, d'abord en utilisant des tables de conversion dédiées aux noms composés, puis en utilisant des tables ne comportant que des noms composés. Pour plus de clarté, ce processus sera décrit à partir de l'exemple de « J.-P.-Ch. », à partir duquel il faut arriver à « Jean-Pierre-Charles ». La reconstruction d'un prénom composé commence par une tentative d'alignement complet avec un prénom composé abrégé présent dans les tables de conversion : un remplacement ne sera fait que si prénom correspond parfaitement aux données présentes dans la table visible en (8). Si il y a alignement complet, alors l'extraction du prénom s'arrête ici. Dans notre exemple, ce n'est pas le cas. Il faut tout de même tenter de reconstruire un prénom, ne serait-ce que partiel. Pour minimiser le risque d'erreur, cette reconstruction partielle se fait en deux temps. Dans un premier temps, l'algorithme cherche à associer une partie des initiales extraites avec un prénom composé abrégé tel qu'il se trouve dans la table de conversion 8. Dans l'exemple, « J.-P. » serait transformé en « Jean-Pierre », puisque « j p » se trouve dans la table de conversion. Si l'intégralité du nom abrégé est reconstruite à l'issue de cette étape, l'algorithme prend fin. Dans notre exemple, il reste à transformer le « Ch. » en « Charles ». L'algorithme essaye donc ensuite d'aligner les initiales restantes avec des clés de la table consacrée aux noms non composés (9). « ch » étant présent dans cette table, le dernier nom abrégé est remplacé par un nom complet : un nom a donc été entièrement reconstruit à partir de son abréviation. Tout au long de ce processus, un système permet de suivre quelles initiales ont été remplacées et lesquelles restent à être traitées, pour éviter de traduire deux fois une seule initiale. À la fin de ce processus, toutes les initiales

qui n'ont pas été remplacées sont supprimées : si les abréviations étaient conservées, cela compromettrait l'obtention de résultats sur l'API. À l'inverse, si seulement une partie des prénoms d'une personne sont extraits, mais que les abréviations sont supprimées, la possibilité d'obtenir des résultats sur l'API augmente.

L'alignement d'un prénom abrégé non-composé avec son équivalent complet est bien moins complexe : l'abréviation d'un prénom non-composé étant plus simple, elle ne demande pas de concevoir d'algorithmes réactifs qui puissent adopter différents comportements en fonction des résultats obtenus. Une fois un prénom abrégé extrait, l'algorithme cherche à l'aligner avec son équivalent complet présent dans la table 9. Si un prénom complet est reconstruit, celui-ci est utilisé pour lancer des requêtes sur l'API de *Wikidata*. Sinon, l'abréviation est supprimée pour ne pas mettre en danger l'obtention de résultats.

Parmi les trois types de prénoms détectés – prénom composé abrégé, prénon non-composé abrégé et prénom complet –, il ne reste donc que les prénoms complets. En théorie, ceux-ci ne demandent pas de retraitement. Cependant, des erreurs peuvent subsister dans les données utilisées. Il est possible que les « marqueurs » d'initiales et d'abréviations (le point à la fin d'un mot commençant par une majuscule) soient absents des documents sources ; le point étant un caractère peu visible, il est également possible que celui-ci soit présent dans les documents d'origine mais qu'il ne soit pas reconnu lors de l'extraction du texte par océrisation. Quelle que soit l'origine du problème, cela mène à des situations comme cela : `<name type="author">MALET (Cl-François)</name>`. Ici, « Cl » est une abréviation ; la structure du nom est cependant celle d'un nom propre composé. La détection de motifs ne pouvant s'appuyer que sur la forme du texte, et non sur sa signification, ce nom est reconnu comme un nom complet. Lorsqu'un prénom est identifié comme étant non abrégé, une étape supplémentaire de vérification a donc lieu : l'algorithme cherche à associer le ou les prénoms avec des prénoms abrégés présents dans les tables de conversions. Si ils se trouvent effectivement dans des tables, alors ils sont remplacés par leur équivalent complet. Sinon, par élimination, les prénoms sont considérés comme n'étant pas abrégés et sont utilisés pour lancer des requêtes sur l'API.

Ce processus de reconstitution des prénoms permet d'augmenter la qualité des données en cherchant à pallier à leur plus grand écueil. Bien que ce processus soit totalement prédictif et qu'il vienne avec un certain risque d'erreur, son rôle est essentiel dans l'obtention de résultats valides sur le moteur de recherche de l'API *Wikidata*, surtout en l'absence d'informations biographiques. Cela apparaît dans les tests menés, et dont le résultat est visible en annexes (1). Lorsqu'une requête est lancée seulement sur le prénom et le nom de famille usuel, l'identifiant *Wikidata* récupéré est valide dans 48% des cas en reconstituant les prénoms. Si des requêtes sont faites sur les noms de famille et les prénoms non reconstitués uniquement, ce taux de réussite tombe à 42%. Cette différence est non négligeable lorsque l'on travaille sur 82000 entrées différentes. Ainsi, en utilisant une approche modulable et adaptée au corpus traité, il est possible de pallier à certaines absences dans les

données disponibles. S'il n'y a pas toujours assez d'informations pour s'assurer que l'identifiant récupéré sur *Wikidata* sera le bon, reconstituer les prénoms permet d'augmenter la précision des données, et donc d'obtenir de meilleurs résultats. Comme cela a été dit plus tôt, cette approche est totalement prédictive, et il est difficile de garantir qu'un prénom reconstruit sera correct. Cette incertitude ne doit pas être oubliée dans la constitution du programme chargé de faire les requêtes en tant que tel : toute opération d'extraction et de reconstitution des données risque d'introduire du bruit. Il est donc important de réaliser les recherches sur l'API de manière, là encore, modulaire, en multipliant le nombre de recherches pour chercher à pallier l'imperfection des données.

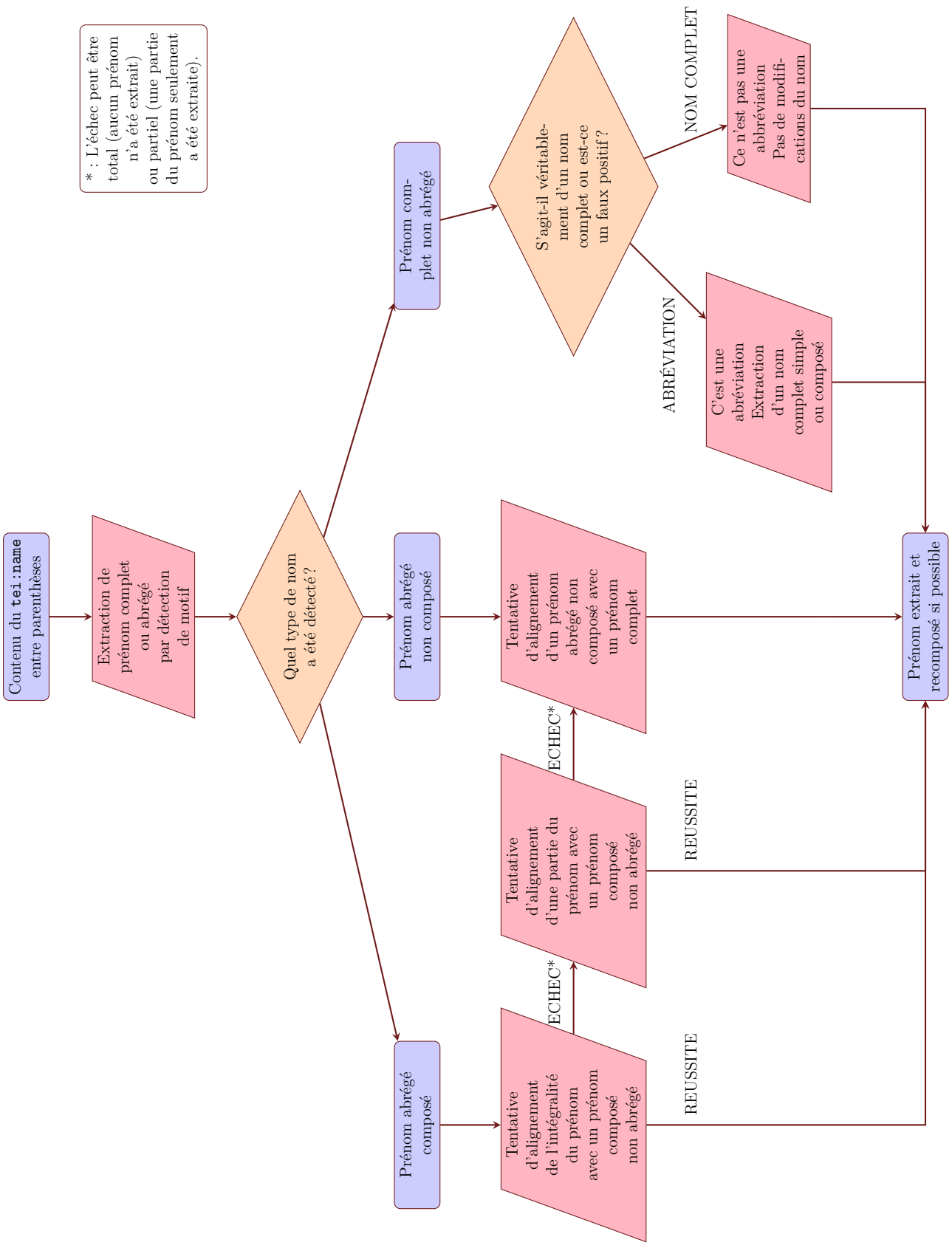


FIGURE 4.5 – Représentation graphique du processus d'extraction et de reconstitution d'un nom à partir de son abréviation

4.4 Extraire des informations biographiques du `tei:trait`

Si les informations nominatives sont déterminantes pour récupérer un identifiant valide, elles sont souvent incomplètes ; la manière dont les informations y figurent ne correspond pas non plus toujours à ce qui est disponible sur *Wikidata*. En plus du processus d'extraction d'informations nominatives issu du `tei:name` présenté ci-dessus, des informations biographiques sont extraites du `tei:trait`. Pour cette étape encore, l'extraction s'appuie sur une détection de motifs augmentée par l'usage de tables de conversion. Deux informations sont récupérées : les dates de vie et de mort – information cruciale car présentant un risque d'erreur très faible – et la fonction occupée par une personne – ce qui permet d'augmenter la certitude que l'identifiant *Wikidata* récupéré pour une entrée de catalogue est valide.

4.4.1 Identifier les dates de naissance et de décès d'une personne

Comparativement à l'extraction d'informations nominatives, l'extraction des dates de naissance et de décès d'une personne par détection de motifs semble assez simple. Là où un nom peut prendre de nombreuses formes et structures différentes, l'expression régulière correspondant à une date correspond à une suite de quatre chiffres, soit :

```
\d{4}
```

S'il est facile de détecter une date, il est moins évident d'être certain que l'année extraite correspond à la naissance ou à la mort d'une personne. En effet, d'autres dates sont souvent présentes dans le `tei:trait`, comme cela apparaît dans l'exemple ci-desous (4.7). Se contenter d'une extraction de toutes les dates du `tei:trait` risquerait de faire exploser la quantité de bruit dans les données utilisées pour faire des recherches sur l'API de *Wikidata*. Comme avec l'extraction d'informations du `tei:name`, détecter des motifs dans le texte ne permet pas d'attribuer une signification à ces motifs – il ne suffit pas de repérer une date pour que cette date correspond à la naissance ou à la mort d'une personne.

Pour comprendre le sens des dates repérées dans le texte, il est encore une fois nécessaire de replacer cette date dans son contexte en s'appuyant sur la nature semi-structurée des documents. En effet, les `tei:trait` ayant tous une structure équivalente et utilisant un vocabulaire limité, il est possible de distinguer les dates pertinentes des autres. Les dates de naissance et de décès sont toujours des compléments dans les phrases, comme cela apparaît dans les différents exemples présentés ci-dessus (4.7). Dans l'exemple 5, par exemple, la date de naissance est un complément de « né » et la date de décès de « fusillé ». C'est donc en identifiant ce à quoi une date se rapporte qu'il est possible de distinguer les dates pertinentes des autres. En général, une date naissance est précédée par « né à, née à ». Le vocabulaire utilisé avec la date de décès d'une personne, quoique plus

```
<!-- exemple 1 -->
<trait>
    <p>maire (en 1848) du XIIe arrondissement, médecin, sous-directeur
    ↪ de l'hôpital de la Salpétrière.</p>
</trait>

<!-- exemple 2 -->
<trait>
    <p>célèbre maréchal de France, vainqueur d'Alger en 1830, né en
    ↪ 1773, mort en 1846</p>
</trait>

<!-- exemple 3 -->
<trait>
    <p>brave général de l'armée de l'Est pendant la guerre de 1870, né à
    ↪ Sarreguemines en 1840, mort en 1876</p>
</trait>

<!-- exemple 4 -->
<trait>
    <p>célèbre révolutionnaire, membre de la Commune de Paris, né en
    ↪ 1838, tué à Rueil le 3 avril 1871.</p>
</trait>

<!-- exemple 5 -->
<trait>
    <p>célèbre jurisconsulte, ministre et sénateur, un des ôtages de la
    ↪ Commune, né à Valence (Drôme) en 1804, fusillé avec l'archevêque
    ↪ de Paris le 27 mai 1871</p>
</trait>
```

Code source 4.7 – Exemples de `tei:trait` contenant des dates autres que les dates de naissance et de mort

coloré, est la encore assez limité : « décapité.e, assassiné.e, fusillé.e, guillotiné.e, tué.e ». Ce vocabulaire étant relativement limité, il n'est pas nécessaire d'utiliser de tables de conversions. Dans le motif permettant d'extraire une date, il suffit simplement de prendre en compte le contexte de celle-ci. Le motif à rechercher correspond donc à :

`Mot indiquant une naissance ou un décès -- autres informations (lieu de naissance...) -- date`

L'implémentation de ce motif sous la forme d'expressions régulières correspond donc à :

pour les dates de naissance : `(^|\s|,|)[Nn](.|\s|ée?).+?(?=\d{4})\d{4}`

pour les dates de decès : `(^|\s|,|)((M|m)|[Mm](\s|orte?)).+?(?=\d{4})\d{4}`
et : `(^|\s|,|)([Dd]écap|[Aa]ssa|[Tt]uée?|[Ff]usi|[Gg]uil).+?(?=\d{4})\d{4}`

Si ce motif est détecté, il est certain que la date est une date de naissance ou de décès. Il suffit alors d'extraire la date de son contexte. Ainsi, sur des documents semi-structurés, il est possible, à l'aide de détection de motifs, de resituer des éléments dans leur contexte afin de produire des données propres et réutilisables.

4.4.2 Identifier l'occupation d'une personne

Identifier et extraire l'occupation d'une personne s'appuie également sur de la détection de motifs, cette fois-ci augmentée de tables de conversion : il existe une grande variété dans les différents types de métiers occupés par une personne ; de plus, un métier tel qu'il est décrit dans un catalogue n'a pas nécessairement d'équivalent sur *Wikidata*. Il est donc nécessaire de normaliser les données extraites.

Le processus d'extraction d'informations à l'aide de tables est assez similaire à l'extraction de titres de noblesse, par exemple : pour chaque `tei:trait` présent dans les catalogues, l'algorithme vérifie si un terme correspond à une occupation à l'aide de la table 1, visible en annexes. Si c'est le cas, l'occupation telle qu'elle est mentionnée dans les catalogues est remplacée par son équivalent normalisé. Celui-ci n'est pas seulement une traduction du terme tel qu'il est présent dans le `tei:trait` : les occupations ou fonctions sont remplacées par une version normalisée, qui permette d'obtenir des résultats sur le moteur de recherche de *Wikidata*. Par exemple, le terme « maréchal » sera remplacé par « marshall » (sa traduction anglaise), tandis que des grades militaires moins élevés (« capitaine », « commandant ») seront remplacés par le terme générique « military ». L'extraction d'informations n'étant pas visée à obtenir des données définitives, il n'est pas nécessaire de préserver les termes et les catégories utilisées dans les catalogues.

Cette extraction d'informations « à l'aveugle » – qui se soucie uniquement des mots utilisés et non du contexte dans lequel il le sont – présente deux problèmes, visibles dans les exemples ci-dessous (4.8).

— Dans le premier exemple, une personne est à la fois désignée comme étant « sculpteur » et « auteur ». À l'aide de la table de conversion, les deux termes sont extraits et normalisés, ce qui donne « sculptor writer » [9]. Non seulement la transformation est fausse (il ne s'agit pas d'un écrivain), mais elle rend inutile l'extraction d'informations : les deux termes se contredisent et ne permettront pas d'obtenir de résultats sur *Wikidata*. Ce problème est récurrent, puisque des personnes ont souvent écrit ou réalisé une œuvre sans qu'elles soient connues comme écrivaines – comme cela apparaît dans le deuxième exemple, où le personnage est général, mais a écrit des

9. « sculpeur écrivain »

mémoires. Il faut donc déterminer laquelle de ces deux fonctions est à retenir pour faire des recherches sur l'API *Wikidata*.

— Au premier problème de l'extraction de plusieurs fonctions s'ajoute une deuxième difficulté : les occupations extraites ne se rattachent pas nécessairement à la personne nommée dans le `tei:name`. C'est le cas dans le troisième exemple : « empereur » est présent dans la table de conversion – contrairement à son équivalent féminin, qui ne permet pas souvent d'obtenir de résultats. La fonction extraite sera donc « empereur / emperor », ce qui fausse les données extraites : c'est la femme de l'empereur qui est décrite dans cette entrée. Il est donc nécessaire de reconnaître lorsque, dans le `tei:trait`, une fonction ne se rattache pas à l'auteur.ice du manuscrit.

```
<!-- exemple 1 -->
<trait>
    <p>célèbre sculpteur, auteur de la statue de la Résistance, érigée à
    ↪ Dijon en 1875 et enlevée par ordre du gouvernement, né à Nuits
    ↪ (Côte-d'Or) en 1815, mort en 1876</p>
</trait>

<!-- exemple 2 -->
<trait>
    <p>célèbre général auteur de Mémoires, qui ont eu un grand succès,
    ↪ n. 1782, m. 1854.</p>
</trait>

<!-- exemple 3 -->
<trait>
    <p>impératrice des Romains, reine de Hongrie, fille de
    ↪ Charles-Quint, femme de l'empereur Maximilien II, morte en
    ↪ 1603</p>
</trait>
```

Code source 4.8 – Difficultés liées à l'extraction de fonctions du `tei:trait`

Les deux problèmes trouvent des solutions différentes. Le premier problème naît de la différence entre les informations présentes dans les catalogues et celles qui sont nécessaires pour l'extraction d'informations de *Wikidata*. Dans un texte fait pour être lu par des humains, il n'y a aucun problème à mettre bout à bout plusieurs occupations, fonctions ou faits notables à propos d'une personne. Un.e lecteur.ice est capable de comprendre la polysémie des mots (« auteur », par exemple) et la relation des différentes fonctions entre elles. Dans le cas d'un catalogue, qui est fait pour vendre, il est normal que les descriptions de personnes soient mélioratives : la multiplication des hauts faits à propos d'une personne met en avant son statut, et justifie donc la valeur du manuscrit vendu. Les techniques de traitement automatique utilisées ne comprennent pas la polysémie. Comme

cela a été dit plus haut, le moteur de recherche de *Wikidata* fonctionne mieux avec des termes spécifiques ; il est moins efficace lorsque le nom d'une personne est recherché avec plusieurs fonctions en même temps. Il est donc nécessaire de déterminer l'occupation principale d'une personne. Après l'extraction des fonctions, les différents termes extraits sont comparés entre eux afin de déterminer le terme à retenir. La comparaison se fait entre les mots qui sont cooccurents, c'est-à-dire qui se retrouvent régulièrement ensemble, comme par exemple « marshall » et « military ». Dans ce cas, c'est la fonction la plus importante et la plus spécifique (« marshall ») qui est conservée, et le terme générique est abandonné. Les termes très génériques – comme le « writer » qui est traduit d'« auteur » – sont presque systématiquement abandonnés si d'autres termes ont été extraits : un.e auteur.ice n'est un.e écrivain.e que si ce terme est le seul utilisé pour le ou la décrire. Sinon, la personne est autrice de quelque chose sans que cela soit sa fonction principale.

Le second problème naît de l'algorithme d'extraction d'informations lui-même : la récupération de fonctions ou de métiers est non-contextuelle : si un terme est présent, il est récupéré. Un post-traitement est donc nécessaire afin de déterminer si le terme se rapporte à l'auteur.ice du manuscrit, ou à une autre personne. L'algorithme cherche à voir si le terme est précédé des prépositions « du, de la, des », comme c'est le cas dans le troisième exemple ci-dessus (4.8). Dans ce cas, il est considéré que l'information récupérée n'est pas relative à l'auteur.ice, mais à une tierce personne et elle est donc abandonnée.

La récupération de données biographiques du `tei:trait` marque la fin du processus d'extraction d'informations ; à l'issue de celui-ci, l'ensemble des éléments récupérés sont stockés dans un dictionnaire structuré (4.2), ce qui permet d'attribuer à chaque terme une fonction, une signification spécifique. À partir d'un texte en langage naturel, il est donc possible d'extraire un certain nombre de paramètres à partir desquels lancer des recherches en plein texte. C'est grâce à la nature semi-structurée des entrées – qui utilisent un vocabulaire peu varié dans des phrases structurellement proches les unes des autres – qu'il est possible de faire une extraction d'informations avec un grand degré de précision en s'appuyant uniquement sur de la détection de motifs. Grâce à une bonne connaissance du corpus et à l'aide d'une approche modulaire, qui s'adapte à différents cas de figure, il est possible d'approximer une compréhension « humaine » du texte. L'extraction d'informations n'étant qu'un étape préalable, cette compréhension du texte est uniquement centrée sur les éléments qui sont pertinents pour le moteur de recherche de *Wikidata*. Cette étape d'extraction est cependant prédictive : elle peut inclure du bruit dans des données, les informations extraites peuvent être mal structurées, puisque leur fonction est déterminée principalement selon leur place dans le texte. De ce fait, l'étape suivante se doit de répondre à cette part de bruit possible, à l'aide d'un algorithme qui soit là encore modulable. L'attribution à chaque terme d'une valeur ou d'une fonction spécifique est mise à profit pour pouvoir construire de façon automatique des recherches intelligentes, qui s'adaptent autant aux données disponibles que à l'outil auquel elles sont destinées :

un moteur de recherche.

Chapitre 5

Résoudre les entités par l'alignement avec des identifiants *Wikidata*

En utilisant les données extraites à l'étape précédente, des recherches en plein texte sont faites sur *Wikidata* de façon automatique. Cette automatisation est permise par l'API développée par l'institution et par un *bot*, c'est-à-dire à un programme informatique chargé de faire des requêtes sur l'API, développé pour l'occasion au sein du projet *MSS / Katabase*. Lorsqu'une recherche est faite, *Wikidata* retourne plusieurs résultats ; il est impossible de vérifier quel résultat est pertinent parmi tous ceux qui sont obtenus, et seul le premier résultat est donc retenu. Celui-ci contient un identifiant *Wikidata*, à partir duquel des requêtes pourront être menées avec `SPARQL` afin de constituer un jeu de données sur les auteur.ice.s figurant dans les catalogues de vente de manuscrits. L'objectif de l'algorithme du *bot* est alors de chercher à maximiser la probabilité que, avec les données produites à l'étape précédente, le premier résultat retourné par l'API soit correct. Pour savoir comment maximiser cette probabilité, il est nécessaire de mieux comprendre la fiabilité des différents paramètres extraits à l'étape précédente. Après avoir étudié l'impact de ces paramètres dans l'obtention d'un résultat valide et après avoir présenté le fonctionnement d'une API, la manière dont l'algorithme a été constitué sera présentée.

5.1 Quantifier l'incertitude : quels sont les paramètres qui permettent d'obtenir le bon résultat ?

Tout au long de la constitution de cet algorithme, des tests ont été menés pour maximiser sa qualité et la probabilité d'obtenir des résultats valides. Le premier test est essentiel à la constitution de l'algorithme : il mesure l'impact de chaque paramètre de recherche (dates de naissance et de décès, titres...) dans l'obtention d'un identifiant valide (1).

Ce test a été mené sur un jeu de 200 entrées sélectionnées dans différents catalogues,

avec une répartition des les différents types de `tei:name` similaire à celle du jeu de données complet ; de plus, la proportion d'entrées avec des `tei:trait` est la même dans le jeu de test et dans l'ensemble du catalogue. Pour chacune de ces 200 entrées, des données sont extraites du `tei:name` et du `tei:trait` ; ensuite, cinq requêtes sont lancées pour chaque entrée : d'abord une requête avec le nom de famille usuel et le prénom seulement ; ensuite, une requête avec ces deux informations et un paramètre en plus – nom de famille noble, titre de noblesse, dates, occupation. Les cinq requêtes sont faites, peu importe si les différentes données sont disponibles pour une entrée ou non : l'intérêt est de pouvoir quantifier l'impact de chaque paramètre pour l'ensemble des entrées, et pas seulement pour celles où des informations ont pu être extraites des catalogues pour un paramètre donné. Par exemple, si les dates de naissance et de décès permettent une obtention du bon résultat dans 90% des cas mais qu'elles sont disponibles pour 5% entrées, leur impact global sur l'obtention d'un résultat valide est en fait très limité. Comme cela a été dit plus haut, ces cinq requêtes sont lancées deux fois : d'abord avec le nom et le prénom complet et reconstitué si besoin, ensuite avec seulement les prénoms non-reconstitués, afin de vérifier l'utilité de la reconstitution (les résultats sont très positifs, avec une augmentation de performance allant jusqu'à +6%).

Ces tests permettent donc de connaître, au niveau de l'ensemble du jeu de données, les paramètres les plus fiables pour obtenir des résultats valides. Il en ressort que le paramètre le plus efficace est l'occupation ou le métier d'une personne : celui-ci permet d'obtenir des résultats valides pour 53,1% des entrées (contre 48% en recherchant les noms et prénoms, et 42% en cherchant les noms et uniquement les prénoms non-reconstruits). Ce résultat peut sembler inattendu puisque les fonctions ne sont pas toutes relevées et qu'elles subissent un retraitement, ce qui pourrait créer du bruit additionnel dans les recherches. Cependant, il montre l'importance de la connaissance à la fois de son corpus et de *Wikidata* : c'est en s'adaptant à son moteur de recherche qu'une utilisation efficace des fonctions a été rendue possible. Le deuxième paramètre le plus efficace est les dates de naissance et de décès (52,6% de résultats valides) : c'est une information qui n'est pas retraduite, et il n'y a pas de risque de création de bruit additionnel. Suivent les noms de famille noble (45,9% de 40,5%, respectivement avec et sans reconstitution des prénoms). Il est intéressant de remarquer que ce paramètre fait diminuer les performances des recherches (entre 1,5 et 2,1 points de moins qu'en utilisant uniquement le prénom et nom de famille usuel). Cette chute s'explique probablement par le fait que, sur *Wikidata*, une personne est rarement nommée à la fois par son nom usuel et par son nom de famille noble. Dans certains cas cependant, rechercher le nom de famille noble à la place du nom usuel peut permettre d'obtenir des résultats qui n'auraient pas été trouvés autrement.

En quantifiant l'impact de chaque paramètre de recherche dans l'obtention d'un résultat, ce test permet de savoir à quelles données se fier et quelles données rechercher en premier sur l'API de *Wikidata*. Il permet donc de quantifier la certitude, ou l'incertitude,

liée à chaque paramètre, et donc de développer un algorithme de recherche de façon plus intelligente. Mais avant de présenter cet algorithme, il peut être intéressant de revenir sur ce qu'est une API, et sur la manière de les utiliser dans un contexte de recherche.

5.2 Automatiser l'extraction de données depuis des sources externes : les API

5.2.1 Qu'est-ce qu'une API ?

Une API Web [1] est un protocole qui permet à un programme d'interagir avec un autre dans le cadre d'une architecture client-serveur. Cette architecture est à la base du Web, qui est formé d'un ensemble d'ordinateurs reliés en réseau. Certains ordinateurs sont des hôtes et servent à diffuser du contenu ou des données : ce sont les serveurs. D'autres ordinateurs sont des clients, c'est-à-dire qu'ils consomment ce qui est fourni par les serveurs [2]. Le client fait des requêtes qui sont traitées par le serveur, qui en retour renvoie une réponse. Ce réseau d'ordinateurs communique grâce à des protocoles, au cœur desquels se trouve le *HyperText Transfer Protocol* (HTTP) [3]. Le rôle de celui-ci est de définir comment un client et un serveur communiquent : il établit la sémantique des requêtes envoyées d'un client à un serveur, et des réponses que ce dernier renvoie en retour. Le protocole HTTP définit donc de façon générale comment deux machines communiquent : lorsqu'un client pose une question, le serveur saura toujours ce que la question signifie ; de la même manière, le client pourra toujours interpréter la réponse du serveur. Cependant, ce protocole ne définit pas ce qui peut être demandé par un client à un serveur ; il ne définit pas non plus précisément le modèle de données utilisé par la réponse du serveur. Des technologies et des protocoles supplémentaires sont donc nécessaires. Ils permettent de définir le contenu et la sémantique des requêtes et des réponses. C'est ici que les API prennent leur importance : elles définissent ce qui peut être requêté, le vocabulaire spécifique à utiliser par les clients et les formats de réponse qui seront fournis en retour. En plus de définir un protocole, elles sont des programmes complets côté serveur : lorsqu'elle reçoit une requête d'un client, l'API traite celle-ci, construit une réponse en interagissant avec serveur et la renvoie à un client. L'API est donc un programme, avec des fonctions qui traitent des données, et un protocole, avec un vocabulaire permettre l'interaction client-serveur. Contrairement au HTTP qui est générique et défini à l'échelle du Web, les

1. Il existe également des API qui ne sont pas faites pour le Web, et qui permettent par exemple la communication entre deux programmes présents sur une même machine.

2. Dans les faits, une seule machine peut à la fois être client et serveur.

3. Roy Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, Version : PDF edition, 1-column for viewing online, Thèse de doctorat, Irvine, University of California, 2000, URL : `https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm` (visité le 05/08/2022), p. 116-127.

API sont spécifiques à un projet, à une application (chaque application ayant des besoins différents) ou à un groupe d'applications [4]. Une API permet donc l'interaction entre deux types de programmes : le programme dit fournisseur (qui est soit l'API en tant que telle, soit un autre programme côté serveur) et le consommateur (c'est-à-dire, un programme côté client) [5]. D'un point de vue de la conception d'applications, les API ont de nombreux avantages. Elles garantissent la sécurité des données : selon les principes REST [6], le client n'accède jamais directement à une base de données dans le serveur, mais seulement à une représentation des données produite à la demande ; puisque les données enregistrées sont distinctes de celles qui sont communiquées, il est également possible de modifier la base de données derrière l'application sans pour autant devoir modifier l'intégralité de l'API ; de la même manière, le programme client n'a pas besoin d'être mis à jour à chaque mise à jour côté serveur : il doit seulement être modifié si l'API est modifiée.

Si l'élaboration d'une API est intéressant d'un point de vue de développement, d'architecture Web et d'interaction entre machines, l'utilisation d'API dans un contexte de recherche en humanités peut être très fructueuse. Elles permettent la communication de données brutes (c'est-à-dire, d'informations « à l'état pur », sans mise en page ou interface graphique) dans des formats documentés. Consommer une API [7] permet donc d'obtenir de façon automatisée des informations déjà structurées, dans un format qui soit compréhensible par une personne humaine autant que par une machine. L'utilisation d'une telle technologie est donc totalement adaptée à l'alignement avec *Wikidata*. Concrètement, en utilisant l'API de *Wikidata*, il est possible de lancer des recherches en plein texte sur le moteur de recherche de cette base de données de façon automatique. Même si les données obtenues doivent être ensuite corrigées, en créant un *bot*, c'est-à-dire un programme pour consommer une API, on s'épargne le travail de faire manuellement plus de 80000 recherches.

5.2.2 Consommer une API : le cas de l'application *MediaWiki*

Wikidata et les projets qui y sont affiliés (*Wikipedia*, *Wikimedia* et autres projets de la *Wikimedia Foundation*) peuvent être consultés en ligne, par des utilisateur.ice.s humain.e.s, au travers d'un site Web traditionnel. Il faut cependant distinguer le site Web des données : le site n'est qu'une interface de présentation. Les projets de la *Wikimedia Foundation* existent avant tout sous la forme d'un ensemble de bases de données (comme le montre le schéma 5), qui sont accessibles au public via le logiciel *open source* MediaWiki [8].

4. Il existe cependant des standards dans l'architecture des API, comme le *Representational state transfer* (REST) , défini par R. Fielding (*Ibid.*, p. 76-106)

5. Pour une représentation graphique de la communication avec une API dans une architecture client-serveur, voir la figure 5.1

6. *Ibid.*

7. C'est-à-dire récupérer des données

8. *Wikimedia Foundation§Technology*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://en.wikipedia.org/wiki/Wikimedia_Foundation#Technology` (visité le 01/08/2022).

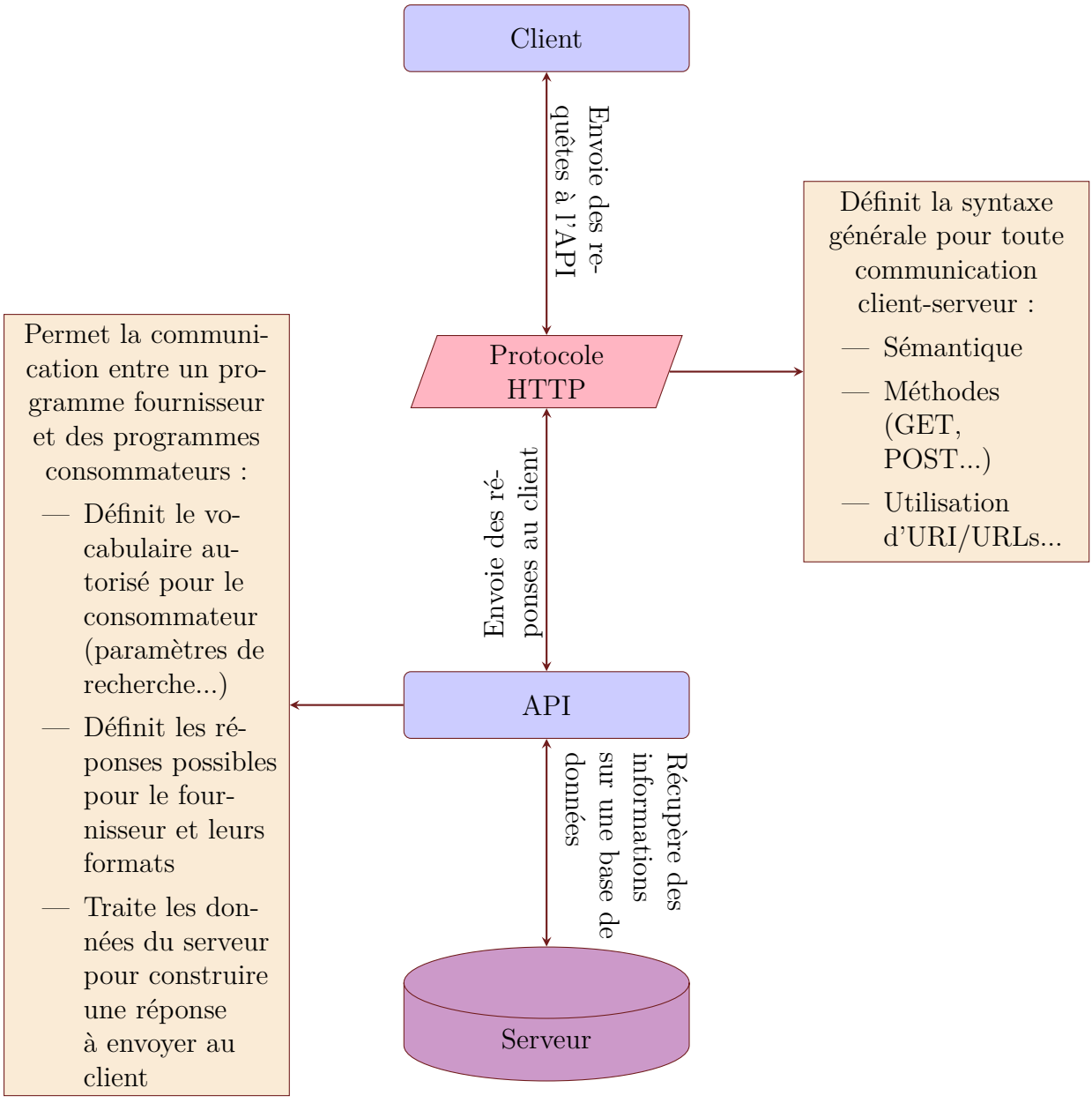


FIGURE 5.1 – L'interaction client-serveur au travers d'une API

Celui-ci permet la diffusion des données aux utilisateur.ice.s via une interface graphique traditionnelle ; mais il est également possible de récupérer et d'envoyer des données de la *Wikimedia Foundation* via l'API de MediaWiki.

Une API Web étant un équivalent d'une application Web conçue pour une machine, la manière d'accéder aux données est assez similaire à celle utilisée pour accéder à un site Web traditionnel : la requête faite par le client prend la forme d'un *Uniform Resource Locator* (URL) [9] analogue à celui d'une page Web traditionnelle. L'URL contient la plupart des informations nécessaires pour que l'API comprenne la requête (interprète ce

9. Dans les faits, dans le protocole HTTP les URL sont nécessaires à une requête, mais ne sont qu'une partie de la requête : elle est également composée d'un en-tête ou encore d'une méthode, c'est-à-dire d'une action à faire avec cet URL (demander des données, en envoyer, en supprimer...).

que le client demande) et construise une réponse (dans notre cas, renvoie des données). Ci-dessous, deux URL correspondant à une recherche en plein texte pour « Virginia Woolf ».

—

Recherche sur le site *Wikidata* :
`https://www.wikidata.org/w/index.php?search=virginia+woolf`

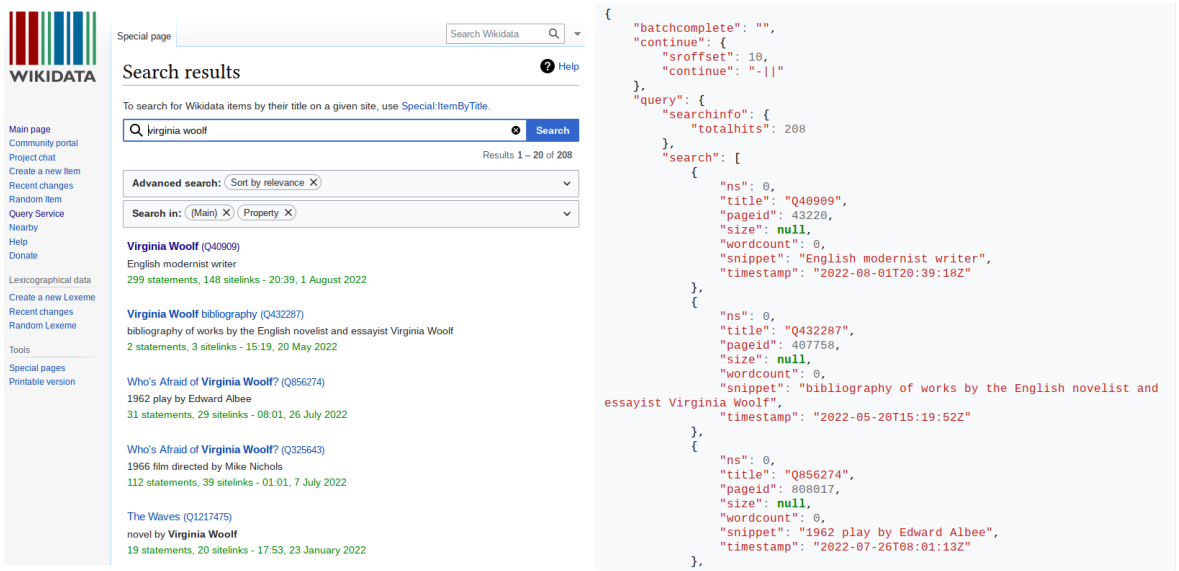
Recherche sur l'API *Wikidata* : `https://www.wikidata.org/w/api.php?action=query&list=search&srsearch=virginia+woolf`

Les deux URL posent la même question ; cependant, le site « pour humain.e.s » est indiqué par `index.php?` ; `api.php?` indique que le deuxième URL renvoie à l'API et contient une suite de paramètres de recherches séparés par le caractère `&` :

— `action=query` : l'action effectuée par cette URL est une requête demandant des données au serveur.

— `list=search` : il s'agit d'une recherche plein texte.

— `srsearch=virginia+woolf` : rechercher les pages qui contiennent « Virginia Woolf ».



(a) Résultats retournés par le site traditionnel

(b) Résultats retournés par l'API

FIGURE 5.2 – Résultats retournés par *Wikidata* pour la recherche « Virginia Woolf »

Enfin, les données récupérées sont les mêmes, mais dans deux formats très différents (5.2). Dans l'image de gauche (5.2a), les résultats sont visibles avec une interface graphique complète, comme sur n'importe quel site Web. À droite cependant (5.2b), les résultats sont présentés sous la forme de données structurées au format `JSON` (format analogue à un dictionnaire). Ce résultat contient à la fois les résultats à proprement parler (en dessous de

« Search ») et des informations descriptives. Par exemple, `query/searchinfo/totalhits` retourne le nombre de résultats pertinents, ici 208.

Cette brève présentation de la manière de consommer concrètement une API indique également le protocole à suivre pour l'alignement des `tei:name` avec des entités *Wikidata* : il s'agit de concevoir un algorithme qui, à partir des données extraites du `tei:name` et du `tei:trait`, construise des URL pour faire des recherches en plein texte sur *Wikidata*. Cela revient à associer au paramètre de l'URL `srsearch` la chaîne de caractère qui doit être recherchée. Une fois qu'un URL a été construit, l'algorithme doit le transmettre à l'API, interpréter le `JSON` renvoyé par celle-ci et enregistrer la réponse dans un fichier. Étant donné que l'alignement avec *Wikidata* prend plusieurs dizaines d'heures et demande des performances élevées, des fichiers de log annexes sont créés en cours d'utilisation pour optimiser l'algorithme.

5.3 Présentation générale

L'algorithme d'alignement avec *Wikidata* construit des URL et lance des recherches en plein texte sur l'API *Wikidata*. De par son fonctionnement, il multiplie les requêtes. Puisqu'il est impossible de vérifier en cours d'exécution si la réponse retournée par l'API contient un résultat valide, il est nécessaire d'organiser l'algorithme afin de maximiser la probabilité d'obtenir un résultat valide. C'est pourquoi les recherches sont faites en commençant par celles qui ont la plus grande certitude d'obtenir des résultats valides et en terminant par les plus incertaines. Dès qu'une réponse est obtenue, l'algorithme s'arrête et l'entrée de catalogue suivante est traitée. Comme cela apparaît dans la figure 5.3, l'algorithme interagissant avec l'API a donc un fonctionnement à plusieurs étapes.

La première étape est la même quelles que soient les informations contenues dans le `tei:name` : l'ensemble des paramètres[10] extraits lors de l'étape précédente sont mis bout à bout afin de lancer une recherche. Cela permet que la recherche soit la plus précise possible, afin de maximiser les possibilités d'obtenir un résultat valide.

Si la réponse de l'API ne contient pas de résultat, alors les données disponibles sont étudiées afin d'adopter le bon comportement. Comme cela a été dit, différents processus ont été utilisés pour extraire des données du `tei:name` et du `tei:trait` ; il est donc logique de ne pas traiter tous les dictionnaires de données de la même manière.

Si l'entrée concerne une personne noble (c'est-à-dire, si le dictionnaire de données contient un nom de famille noble), alors il y a un possible conflit dans les données extraites : si le nom de famille usuel est recherché en même temps que le nom de famille noble sur *Wikidata*, il arrive qu'aucun résultat ne soit retourné. Dans ce cas, une seconde requête

10. Dans ce qui suit, le terme « paramètre » se réfère aux clés du dictionnaire d'informations structurées, et non aux paramètres de l'API présentées ci-dessus (`srsearch`...). Un paramètre correspond donc à un type d'information à rechercher sur l'API : prénom, nom de famille, fonction...

est faite en supprimant le nom de famille noble ; si aucun résultat n'est obtenu, celui-ci est rajouté et c'est le nom de famille usuel qui est enlevé. Pour finir, une autre recherche est faite sans le prénom. Si à ce stade aucun résultat n'a été obtenu, alors un algorithme de recherches soustractives s'exécute ; celui-ci est décrit plus bas.

Le deuxième cas de figure concerne la reconstitution des prénoms. Bien que celle-ci augmente la proportion de résultats valides, un prénom mal reconstitué risque d'empêcher d'obtenir un résultat sur l'API. C'est pourquoi une clé `rebuilt` a été ajoutée au dictionnaire qui contient les données extraites : elle permet d'indiquer qu'un nom a été reconstitué. Dans ce cas, une nouvelle recherche est faite sans le prénom. Si aucun résultat n'est obtenu, alors c'est l'algorithme de recherches soustractives qui s'exécute.

Le troisième cas de figure concerne la présence de plusieurs paramètres de recherche disponibles, c'est-à-dire de plusieurs valeurs non-nulles dans le dictionnaire. Dans ce cas, c'est directement l'algorithme soustractif qui s'exécute. Celui-ci a été conçu à partir du constat que, d'un côté, les données contiennent inévitablement de bruit ; de l'autre, l'extraction de données risque elle aussi d'introduire du bruit. Cette logique soustractive cherche donc à supprimer le bruit qui a pu s'introduire. Afin d'éviter d'être trop « brut », l'algorithme procède en séparant les différents types de bruit possibles. D'abord, il arrive souvent qu'un chiffre soit mal reconnu lors de l'océrisation, et il peut donc y avoir des erreurs dans les dates. Deux dates peuvent être extraites (naissance et décès) ; il y a cependant peu de chances que ces deux dates soient erronées. Par conséquent, une des deux dates est supprimée et la recherche est faite ; si aucun résultat n'est obtenu, c'est l'autre date qui est supprimée. La seconde source de bruit provient des prénoms : comme cela a été dit, au XIX$^{\text{ème}}$ s., les prénoms étrangers sont régulièrement traduits ; il arrive souvent que les prénoms composés ne soient pas dans le même ordre que sur *Wikidata*, ou qu'une personne y soit nommée par son deuxième prénom – peut-être parce que les catalogues utilisent des noms d'usage alors que *Wikidata* s'en tient aux noms civils. Pour ces raisons, le prénom est retiré de la chaîne de caractères à rechercher avant de faire une requête sur l'API. Bien que les risques liés aux dates et aux prénoms ont été corrigés (autant que possible), il est possible que l'auteur.ice n'ait pas encore été aligné avec une entité *Wikidata*. À ce stade, tous les risques spécifiques ont été traités. L'algorithme relance alors des recherches en retirant à chaque fois un paramètre différent, jusqu'à ce qu'un identifiant ait été extrait de *Wikidata* ou que toutes les permutations possibles aient été faites.

Cette série de constructions d'URL cherche à minimiser les redondances ; cependant, il est possible qu'une même chaîne de caractères puisse être extraite à différentes étapes. C'est pourquoi des fichiers de log sont créés en cours d'utilisation : ils servent à sauvegarder les chaînes de caractères qui ont déjà été recherchées et les résultats qui leur sont associées. Avant de lancer une requête sur l'API, l'algorithme vérifie donc si la chaîne a déjà été recherchée. Si oui, il récupère le résultat obtenu ; sinon, lance la recherche et sauvegarde le résultat retourné par *Wikidata*.

L'algorithme de recherche sur *Wikidata* est visiblement complexe, puisque son objectif est de minimiser le taux d'erreur possible sur des données variées, ce qui demande d'adopter des comportements différents en fonction des informations disponibles. L'algorithme ayant tendance à retirer de plus en plus de paramètres de recherches en cours d'exécution (les dates, les prénoms...), il est possible que la chaîne recherchée finisse par être très pauvre en informations ; cela permet de s'aligner avec une entité, mais celle-ci risque de ne pas correspondre au `tei:name` ou d'être très générique. Si l'entité recherchée est « Napoléon Bonaparte », mais de nombreux de paramètres ont été retirés, il est possible que l'algorithme ne recherche que « Bonaparte ». Le résultat obtenu est alors « Charles Lucien Bonaparte ». Il est donc utile d'avoir une mesure de la certitude (ou l'incertitude) de la validité d'un résultat, ne serait-ce que pour accélérer le processus de correction des identifiants (ce qui, sur plus de 82000 entrées, n'est pas une mince affaire). Le parti pris a été, tout simplement, d'établir un score de certitude en fonction du nombre de paramètres utilisés pour lancer une recherche. Ce score a été défini de façon expérimentale, en mesurant la proportion de résultats considérés comme certains et ainsi que le taux d'erreur au sein des résultats certains (c'est-à-dire, le nombre de résultats qui ont dépassé le seuil de certitude mais qui sont en fait erronés). Après plusieurs essais différents, le score de certitude c_r pour un résultat r a été déterminé. Il suit la formule ci dessous :

Étant donnés :

c_r le score de certitude pour un résultat r ;

q_r la chaîne de caractère recherchée pour obtenir le résultat r ;

$\sum param$ la somme des paramètres utilisés dans q_r ;

$d = 1$ si une date est dans q_r ; sinon, $d = 0$;

$p = 1$ si le prénom utilisé dans q_r n'a pas été reconstitué ; sinon, $p = 0$;

$$c_r = \sum param + d + p$$

Un résultat est considéré comme étant « certain » si $c_r >= 4$ ou si des dates sont présentes dans q_r (les chances qu'un alignement soit fait entre un `tei:name` et une entité homonyme ayant les mêmes dates de naissance ou de décès étant très faibles). Sur le jeu de données de test, 32% des entrées atteignent ce seuil de certitude. 23,6% de celles-ci comprennent en fait une erreur, soit 8,5% du jeu de données total (voir le tableau en annexes : 2). L'objectif de ce score de certitude n'est pas d'éliminer toutes les incertitudes, mais de réduire celles-ci à un niveau acceptable pour accélérer la relecture des résultats : en acceptant un taux d'erreur total de 8,5%, il est possible ne corriger que les entrées pour lesquelles $c_r < 4$. Seuls les deux tiers du jeu de données restent alors à corriger manuellement.

Cet algorithme cherche à répondre à plusieurs problèmes techniques : il travaille avec des données historiques contenant du bruit et un niveau très inégal d'information. Son fonctionnement vise à maximiser la probabilité d'obtenir un résultat valide, en faisant des requêtes d'abord très spécifiques puis de plus en plus génériques. Cependant, il crée lui-même plusieurs problèmes : son fonctionnement est assez complexe et demande de manipuler plusieurs lourds fichiers en même temps. Un ensemble d'optimisations ont donc été réalisées.

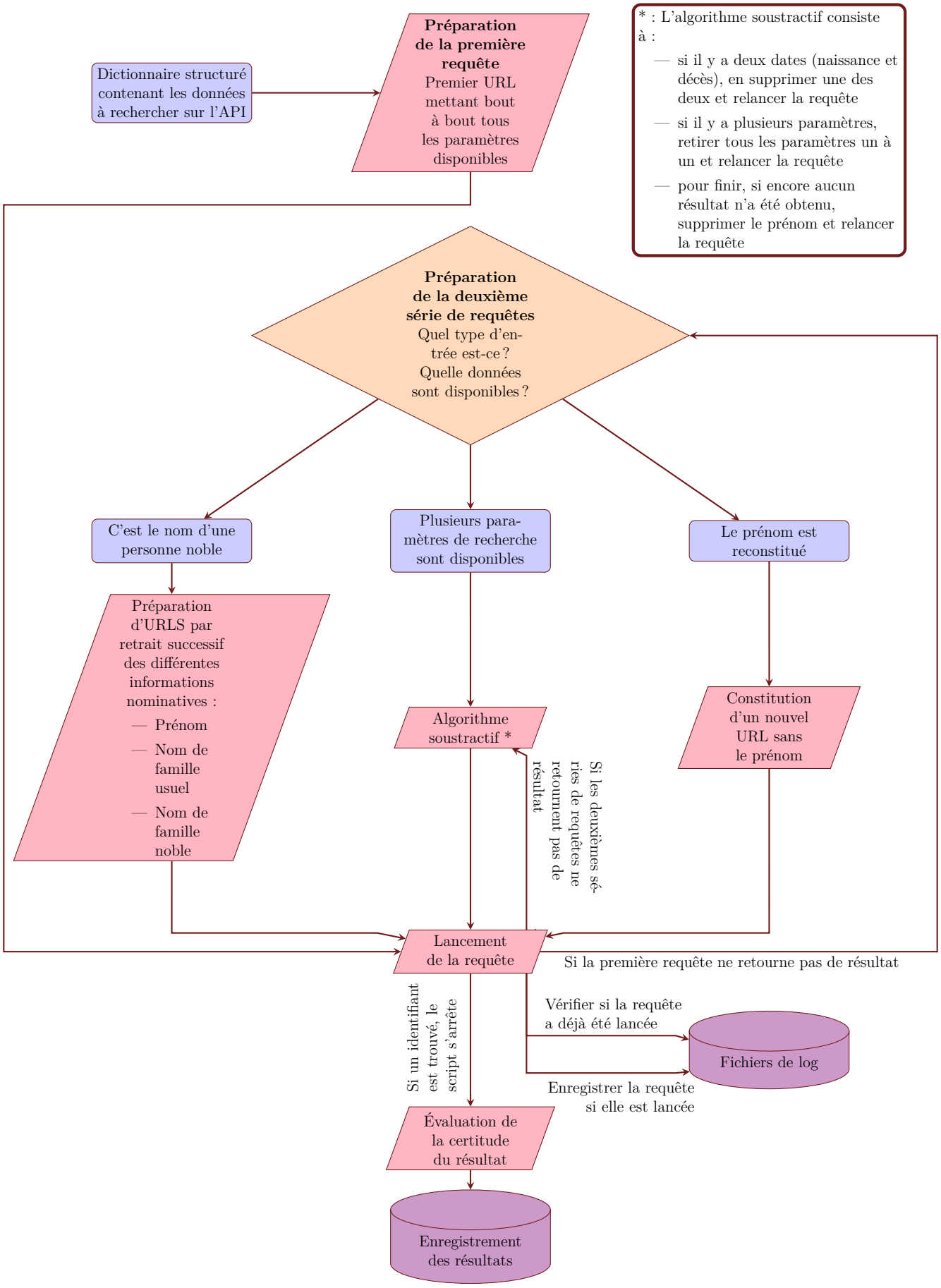


FIGURE 5.3 – Algorithme lançant des recherches en plein texte sur *Wikidata*

5.4 Gérer la montée en charge : optimisation et réduction du temps d'exécution

L'intégralité de l'algorithme (contenant extraction d'informations et interaction avec une API) a cessé de fonctionner sur mon ordinateur alors qu'à peine 5% des 82000 entrées avaient été traitées. Ce problème technique soulève des difficultés adjacentes à la recherche en humanités numériques : dans ce contexte, des machines très performantes ne sont pas toujours disponibles, et la recherche se fait souvent des ordinateurs de bureau traditionnels. De telles situations, la recherche peut parfois faire face à des limitations matérielles. De telles limites sont – bien sûr – un problème ; en obligeant chercheur.euse.s à intégrer des contraintes matérielles dans leurs algorithmes, cette difficulté peut cependant aider à réaliser des programmes plus efficaces, moins gourmands en ressources et en énergie, et donc aider à faire de la recherche plus durable. De telles considérations écologiques sont partagées par toute une communauté de chercheur.euse.s issu.e.s des humanités et défendant des approches « low-tech » [11] ; ces considérations écologiques ne se limitent cependant pas au champ des humanités et se retrouvent dans des travaux provenant de l'informatique « traditionnelle » et de l'apprentissage machine [12]. L'optimisation de programmes informatiques n'est pas seulement une question écologique : c'est également une manière d'écrire de meilleurs programmes, plus performants et efficaces.

Dans sa première version, le script d'alignement avec *Wikidata* avait un fonctionnement apparemment plus « simple », puisqu'il ne prenait pas du tout en compte les questions d'optimisation : pour chaque entrée de catalogue, des informations étaient extraites et les recherches faites sur *Wikidata*. Ce fonctionnement avait deux défauts. D'abord, il demandait que tout le script soit exécuté en une fois – si il y a une erreur imprévue, il est nécessaire de recommencer à nouveau. Ensuite, une interaction avec *Wikidata* était nécessaire pour chaque entrée, peu importe si le nom contenu dans le `tei:name` avait déjà été recherché auparant (alors qu'une même personne est souvent l'auteur.ice de plusieurs manuscrits recensés dans les catalogues) ; l'interaction avec *Wikidata* prenant plusieurs secondes, cela peut grandement ralentir le processus d'alignement avec la base de connaissances. C'est à partir de ces deux problèmes que le processus d'optimisation a commencé.

11. Le *Minimal computing* (« Informatique minimale » en français) est un groupe de réflexion créé au sein de *GO :: DH – Global Outlook Digital Humanities*. Il s'intéresse aux manières de rendre l'informatique minimale, dans différents sens du terme : à la fois plus accessible, mais également plus écologique. Plus généralement, le courant du *minimal computing* développe une approche réflexive et politisée sur les façon de faire de la recherche durable et libre (au sens de l'*open source*) ; ce groupe s'intéresse également au développement des humanités numériques dans un contexte non-occidentalJentery Sayers, *Minimal definitions*, Minimal Computing - a working group of GO ::DH, 2 oct. 2016, URL : `http://go-dh.github.io/mincomp/thoughts/2016/10/02/minimal-definitions/` (visité le 10/06/2022).

12. Emma Strubell, Ananya Ganesh et Andrew McCallum, « Energy and Policy Considerations for Deep Learning in NLP », *CoRR*, abs/1906.02243 (2019), DOI : `https://doi.org/10.48550/arXiv.1906.02243`.

Le premier problème – l'impossibilité de vérifier quelles entrées de catalogue ont déjà été traitées – a d'abord été corrigé en consultant, à chaque fois qu'un nouvel identifiant était recherché, le fichier de sortie. Celui-ci est un tableur qui contient l'identifiant des entrées, le `tei:name`, le `tei:trait`, l'identifiant récupéré sur *Wikidata* et un extrait de la description associée à cet identifiant. Si cette solution permet de ne pas recommencer à zéro à chaque fois que le script est lancé, elle demande de charger en mémoire un très large fichier `csv` (12,7 mégaoctets). Cette vérification particulièrement gourmande a été remplacée par la création d'un fichier de log annexe en cours d'exécution. Celui-ci contient uniquement les identifiants des différentes entrées de catalogue et mesure seulement 1,3 mégaoctets. À chaque fois qu'une nouvelle entrée est traitée, ce fichier est ouvert, et le script vérifie que le nouvel identifiant ne s'y trouve pas. Si l'identifiant ne s'y trouve pas, alors le `tei:name` et le `tei:trait` de cette entrée sont traités et alignés avec un identifiant *Wikidata*. À la fin de ce processus, l'identifiant `TEI` de l'entrée est ajouté au fichier de log. Cette simple modification permet donc de vérifier si une entrée de catalogue a déjà été traitée à l'aide d'un fichier dix fois moins lourd que dans la tentative précédente.

Ce processus permet de vérifier quelles entrées ont déjà été traitées, mais il ne permet pas de savoir si une personne se retrouve d'une entrée sur l'autre. Si une telle récurrence est identifiée, il serait possible de récupérer les résultats déjà disponibles, plutôt que de réaliser à nouveau l'alignement avec *Wikidata*. Non seulement cela pourrait rendre le script plus rapide – le lancement des requêtes pour une entrée prenant environ 2 à 5 secondes –, mais en plus, cela éviterait de surcharger l'API de requêtes, ce qui, dans un cas extrême, peut faire saturer une application en ligne et perturber son fonctionnement. La difficulté consiste alors à désambiguïser les informations présentes dans le `tei:name` (c'est-à-dire, à repérer toutes les occurrences d'une même personne dans les `tei:name`). Cette désambiguïsation est compliquée (l'un des intérêts de la résolution d'entités nommées est justement d'identifier les multiples occurrences d'une même personne). Plutôt que de trouver un moyen de désambiguïser réellement les entrées, une solution plus simple a été mise en place : chaque chaîne de caractère recherchée est enregistrée dans un fichier de log, avec l'identifiant *Wikidata* récupéré par recherche. L'intégralité du processus d'extraction d'informations des `tei:name` et `tei:trait` est donc menée sur toutes les entrées ; l'ensemble des chaînes de caractères à rechercher sont également construites ; cependant, une recherche n'est faite sur l'API que si elle n'a jamais été faite auparant. Du fait de la taille du jeu de données, un problème apparaît cependant : plusieurs chaînes de caractères étant construites pour chaque entrée de catalogue, l'utilisation d'un seul fichier de log pour contenir toutes ces recherches amènerait à la création d'un immense fichier (plusieurs centaines de milliers de lignes). Il serait alors nécessaire de le parcourir intégralement à chaque fois qu'une requête est lancée ; le script optimisé risquerait alors d'être moins efficace qu'avant son optimisation. Le choix a donc été fait de créer automatiquement et d'utiliser plusieurs fichiers de log : les chaînes de caractères y sont enregistrées en

fonction du premier caractère qu'elles contiennent, avec un fichier de log par lettre. Ainsi, avant de lancer une recherche, le fichier à parcourir est bien plus court (2060 entrées pour la lettre « e »).

Avec toutes ces optimisations, le script a pu fonctionner en utilisant au maximum 3 gigaoctets de mémoire vive, soit l'équivalent ce qui est utilisé pour naviguer sur internet en ayant un logiciel de traitement de texte ouvert (sur ma machine). Cependant, l'utilisation fichiers de log faite ici – qui a elle même été optimisée – demande de faire un arbitrage. En n'enregistrant pas de données et en lançant des requêtes à chaque fois, c'est la charge processeur qui augmente, ainsi que la consommation de courant due à internet. En enregistrant les requêtes déjà lancées, ces charges là sont diminuées, mais l'utilisation de mémoire vive augmente (il faut charger en mémoire et lire de larges fichiers). De ces deux deux différentes approches (augmenter la charge processeur ou augmenter la mémoire vive) il est nécessaire de déterminer laquelle est la plus efficace. C'est un des objectifs des tests menés sur l'algorithme d'alignement avec des identifiants *Wikidata*, comme nous allons le voir maintenant.

5.5 Évaluer la résolution d'entités : performance, qualité des données extraites de *Wikidata* et comparaison avec d'autres projets

Le processus d'alignement avec des entités *Wikidata* n'est pas parfait, et implique nécessairement un taux d'erreur. Au vu de la taille du corpus, la correction des résultats est un processus particulièrement chronophage. Pour pouvoir utiliser les données produites avant cette correction, il est nécessaire de trouver une manière d'évaluer la qualité des résultats. Des tests ont été réalisés pour faire cette évaluation. Ils ont également été utilisés durant tout le développement de ce processus, afin de comparer les différents algorithmes d'extraction et de requêtes entre eux. Les tests ont également permis de mesurer l'impact de chaque paramètre de recherche dans l'obtention d'un résultat valide. Enfin, l'évaluation des algorithmes permet de comparer sa version optimisée et non-optimisée.

Comme cela a déjà été dit, les tests ont été lancés sur un jeu de données représentatif contenant 200 `tei:name` et `tei:trait`, choisis dans différents catalogues ; ce jeu de test représentant également les différents types de noms possibles (de personnes nobles, de lieux...) ; enfin, la proportion de `tei:trait` dans le jeu de données complet et dans celui de test est la même. Deux catégories de tests ont été réalisés. Le premier, décrit plus haut, concerne l'impact de chaque paramètre dans l'obtention d'une entité correcte sur *Wikidata* (1). Le second sert à mieux identifier les performances de l'algorithme final (2).

Ce dernier test réalise l'intégralité du processus d'extraction d'informations et d'alignement avec *Wikidata* sur l'ensemble du jeu de test. Ce processus est réalisé trois fois

sans utiliser des fichiers de log et trois fois en les utilisant (c'est-à-dire, en inscrivant les recherches déjà faites dans un fichier et en vérifiant avant chaque recherche si une chaîne de caractère a déjà été recherchée). Faire plusieurs fois de suite le même processus permet de les variations de performance induites par l'utilisation de fichiers de log. La performance d'un script, en terme d'utilisation des processeurs et de la mémoire vive, est assez difficile à quantifier ; elle peut également varier d'une machine à une autre ; enfin, il n'est pas possible de mesurer la charge supplémentaire infligée à un serveur distant lors de l'utilisation répétée de sources de données en ligne. Le seul critère retenu a donc été le temps d'exécution. Le script étant lancé trois fois en utilisant des fichiers de log et trois fois sans, les deux temps d'exécution $\bar{t}$ correspondent au temps moyen nécessaire à traiter les 200 entrées du jeu de test. Soit t_i le temps pris pour traiter une fois le jeu de test, la moyenne $\bar{t}$ correspond à :

$$\bar{t} = \frac{1}{3}\sum_{i=1}^{3} t_i$$

La difficulté pour mesurer les temps d'exécution t_i en utilisant des fichiers de log est que, en dehors des tests, ces fichiers sont de plus en plus volumineux ; le temps utilisé pour les parcourir augmente en conséquence. Pour que ces t_i soient représentatifs, de faux fichiers de log sont créés avant le lancement des tests : ceux-ci comprennent des fausses données qui correspondent environ aux résultats obtenus pour 30000 recherches lancées sur l'API (soit un peu moins que les 82000 entrées du jeu complet, en prenant en compte que certaines recherches ne retournent pas de résultat, ce qui allège d'autant le fichier à parcourir). Le temps d'exécution $\bar{t}$ en utilisant des fichiers de log est de 88,3 secondes ; sans ces fichiers, $\bar{t}$ est de 92,5 secondes. Si la différence entre les deux résultats n'est pas immense, l'utilisation de fichiers de log permet également de limiter la charge processeur et l'interaction avec *Wikidata*, et permet donc d'autres améliorations au delà du seul temps de performance.

À part cette seule évaluation de performance, les tests sur l'algorithme final (2) servent à mesurer la qualité des résultats obtenus. La mesure de performance pour un processus d'alignement est le score F1, qui prend à la fois en compte la précision (le nombre de vrais positifs obtenus parmi tous les résultats obtenus) et le rappel (le nombre de vrais positifs obtenus parmi tous les vrais positifs). Celui-ci est de 67,4%, ce qui marque une nette amélioration (près de 20 points) de l'utilisation seule du prénom et du nom de famille usuel extraits du `tei:name`.

Ce score est également intéressant en comparaison avec d'autres projets de NEL. Aicha Soudani *et al.* ont, en 2018, présenté les résultats d'un projet de résolution de lieux sur un corpus relativement proche de clui de *MSS / Katabase* : des textes littéraires du XIX$^{\text{ème}}$ s. encodés en `XML-TEI` [13]. Les méthodes utilisées par ce projet diffèrent largement

13. Aicha Soudani, Yosra Meherzi, Asma Bouhafs, F. Frontini, C. Brando, Yoann Dupont et Fré-

de celles présentées ici : les auteur.ice.s s'appuient fortement sur l'apprentissage machine, en identifiant d'abord des entités dans le texte à l'aide de `SEM` avant de lier les entités à différentes bases de connaissances en utilisant `REDEN`. Leur projet présente donc une complexité supplémentaire, puisqu'il demande de repérer les entités, ce qui n'est pas le cas ici (elles se trouvent toutes dans des balises `tei:name`). Cependant, c'est cette deuxième étape qui nous intéresse ; les entités ont déjà été identifiées, ce qui nivelle cette différence entre les projets. Trois bases de données en ligne ont été utilisées : *DBPedia*, *DataBnF* et *Wikidata*. En comparaison avec ce projet, il est intéressant de remarquer que le processus d'alignement avec *Wikidata* développé par *MSS / Katabase* obtient un score F1 très satisfaisant (67,4%). Celui-ci dépasse les scores obtenus par Soudani *et al.* dans leur alignement avec *Wikidata* (61,1%) et *DBPedia* (53,6%) (voir le détail des résultats en annexe : 3) [14]. Les scores comparativement positifs du projet *MSS / Katabase* sont peut être également rendus possibles par la quantités d'informations disponibles sur chaque personne dans le corpus, récupérée grâce à un travail d'extraction des données sur l'ensemble du `tei:trait` et du `tei:name`.

5.6 En conclusion : retour sur l'extraction d'informations des catalogues et sur l'algorithme d'alignement avec *Wikidata*

L'algorithme liant les noms d'auteur.ice.s dans les catalogues avec des entités *Wikidata* propose une réponse à plusieurs difficultés techniques propres à la reconnaissance d'entités par la détection de motifs sur un corpus de données varié. Parmi toutes ces difficultés possibles, les trois les plus importantes sont le bruit dans les données d'origine, le bruit produit par l'extraction de données et enfin la difficulté à réaliser un alignement entre des documents historiques et des bases de données contemporaines. La solution proposée s'appuie sur un fonctionnement modulaire, qui s'adapte aux données produites et prend en compte la possibilité que du bruit se soit introduit à une étape ou à une autre. De la différence de résultats obtenue avec Soudani *et al.*, il faut retenir que la technique utilisée n'est pas toujours garante de la qualité des résultats. Plutôt que de chercher à établir des correspondances exactes, l'approche décrite dans ce mémoire, basée sur une détection de motifs *low-tech* a permis de développer une méthode très précisément adap-

dérique Mélanie-Becquet, « Adaptation et évaluation de systèmes de reconnaissance et de résolution des entités nommées pour le cas de textes littéraires français du 19ème siècle », dans *Atelier Humanités Numériques Spatialisées (HumaNS'2018)*, Montpellier, 2018, URL : `https://hal.archives-ouvertes.fr/hal-01925816` (visité le 10/08/2022).

14. Il est à remarquer que Soudani *et al.* disposent d'un score d'« exactitude totale du liage » (*Overall linking accuracy*) qui est nettement plus élevé (entre 70 et 85%) ; la méthode de calcul de ce score, qui mesure la performance de l'ensemble du processus de NEL, n'est cependant pas décrite dans l'article. Il est donc impossible d'y comparer les résultats de *MSS / Katabase*.

tée à la fois au corpus et à *Wikidata*. Ce qui est potentiellement perdu par la détection de motifs est donc corrigé par l'algorithme menant des requêtes sur *Wikidata*.

L'alignement avec des entités *Wikidata* est la meilleure manière de corriger le bruit dans les données du projet : en liant les différents `tei:name` à des identifiants *Wikidata*, il devient possible d'extraire des données structurées d'une source externe, pour construire une base de connaissances plus fiable : même si un corpus semi-structuré facilite le travail d'extraction d'informations, l'accès aux informations reste relativement complexe ; il existe de plus des biais dans les informations contenue par le corpus (ce qui est mis en avant dans les `tei:trait` est avant tout ce qui est susceptible d'intéresser le public de l'époque). En se servant de cette résolution d'entités pour constituer une base de connaissances sur mesure, il est possible de chercher à corriger le biais propre à toute source de données, tout en rassemblant assez d'informations pour pouvoir faire une analyse des facteurs déterminant le prix d'un manuscrit sur le marché du XIX$^{\text{ème}}$ s.. C'est la manière dont est constitué le jeu de données issu de *Wikidata* qui est expliqué ci-dessous.

Chapitre 6

Après l'alignement, l'enrichissement : utiliser `SPARQL` pour produire des données structurées

L'alignement des noms contenus dans les `tei:name` avec *Wikidata* n'est qu'une étape, et non la fin en soit du processus – bien que ce soit la partie la plus difficile. Les identifiants ainsi récupérés servent de « porte d'entrée » aux données disponibles sur la base de connaissances : une fois que ces identifiants sont connus, il est possible de récupérer automatiquement via `SPARQL` des données sur les auteur.ice.s et autres entités présentes dans les catalogues. Obtenir ces informations permet avant tout de mieux situer l'auteur.ice d'un manuscrit, afin mieux le statut d'une personne et de comprendre quelles informations biographiques influencent le prix d'un manuscrit. Mais le liage d'entités nommées peut avoir d'autres fins et permettre d'enrichir le corpus de catalogues de plusieurs manières, comme nous le verrons.

6.1 Comprendre les particularités des modèles sémantiques de données

Un bref retour sur la manière dont s'organisent les données contenues dans *Wikidata* permet de mieux comprendre le problème que peut poser la diversité du corpus pour la constitution d'une base de connaissances propre au projet. *Wikidata*, comme de nombreuses autres bases en lignes (*DataBnF, DBPedia...*), permet d'accéder à des informations stockées dans une base de données en graphe. Ce type de base de données a un fonctionnement très particulier qui influence la constitution d'une base de connaissances à partir des identifiants *Wikidata* récupérés à l'étape précédente. Dans une base de données

en graphe, les données sont souvent encodées en `XML-RDF`. Ce format – dit sémantique – contient des données liées sous forme de « triplets » sujet–prédicat–objet, où :

— le sujet est la ressource principale.

— le prédicat est une propriété du sujet, qui caractérise une relation avec une autre ressource, l'objet.

— l'objet est une ressource secondaire : c'est la valeur d'un prédicat.

Le principe des triplets `RDF` est mieux exprimé sous forme graphique (6.1) :

FIGURE 6.1 – Exemple de relation sujet – prédicat – object

Trois particularités supplémentaires définissent les formats sémantiques :

— Toutes les « ressources » peuvent être tour à tour sujet ou objet. L'exemple du dessus, par exemple, aurait pu être réécrit sous la forme : *Les porteuses* a été peint par Natalia Gontcharova. Dans ce cas, Natalia Gontcharova est l'objet et *Les porteuses* est le sujet. Par conséquent, une base de données en graphe peut être représentée sous la forme d'un réseau de ressources qui entretiennent des relations bilatérales entre elles. Il n'y a pas de hiérarchie entre les informations, contrairement à une base de données `XML` classique.

— L'ensemble des ressources et des prédicats d'une base de données en graphe sont définis et disposent d'un identifiant unique (certains de ces identifiants ont été récupérés grâce au liage d'entités nommées).

— Si le langage `SPARQL` offre une syntaxe commune, chaque base de connaissances peut utiliser des vocabulaires particuliers qui sont organisés en « ontologies ». Une ontologie correspond à la définition d'un ensemble de catégories, de propriétés et de relations qui unissent des données et des concepts ; cet ensemble est complété par une modélisation (souvent sous forme graphique), qui indique la relation entre les différents termes de l'ontologie [1]. Ceux-ci sont souvent liés de façon hiérarchique (plusieurs termes spécifiques pouvant être dérivés d'un terme générique). Au sein d'une ontologie, chaque terme a lui-même un identifiant unique, ce qui garantit une implémentation uniforme pour l'ontologie. Une même base de connaissances peut utiliser plusieurs ontologies. Celles-ci sont définies à l'aide d'espaces de noms [2],

1. *Ontology (information science)*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://en.wikipedia.org/wiki/Ontology_(information_science)` (visité le 01/02/2022).

2. *Namespace*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://en.wikipedia.org/wiki/Namespace` (visité le 05/05/2022).

c'est-à-dire par des identifiants qui permettent de différencier les ontologies. Les prédicats, plus particulièrement, sont définis selon une ontologie particulière.

SPARQL a l'avantage de permettre de récupérer des données structurées sur des bases en ligne et offre une syntaxe unique partout où il est implémenté. Cependant, le troisième point complexifie son utilisation, ainsi que la définition des données à récupérer : les prédicats sont décrits avec une grande précision et selon des vocabulaires spécifiques ; par conséquent, une plusieurs relations sémantiquement analogues peuvent être représentée par différents prédicats selon le type de donnée qui fait l'objet d'une requêtée (une personne, une sculpture...). Dans l'ontologie *Wikidata*, la création d'un texte et la création d'une peinture ne correspondent pas au même prédicat, bien qu'il s'agisse dans les deux cas de la création d'une œuvre. Pour que les données soient utilisables, il faut être très spécifique quant aux informations recherchées.

6.2 Quelles données rechercher via SPARQL ?

La question des données qui doivent être récupérées, et donc de la base de connaissances à constituer pour le projet *MSS / Katabase* n'est pas anodine. D'abord, l'utilisation de sources externes peut être utilisée pour corriger des biais dans les données originelles (où ce sont les auteur.ice, éditeurs et éditrices de catalogues qui décident des informations à intégrer). Ensuite, si l'objectif premier est de récupérer des données pour mener une étude économétrique, l'alignement avec *Wikidata* peut également permettre d'autres enrichissements et fonctionnalités. Enfin, cette étape représente une difficulté technique, du fait de la spécificité de SPARQL. Il faut donc construire des requêtes très étendues, afin d'obtenir des résultats pour tous les types de données. Les 18899 entités avec lesquelles les entrées de manuscrits ont été alignées peuvent se classer en de nombreuses catégories. Sur *Wikidata*, une entité est une « instance » d'une classe plus large. C'est en fonction de ces classes qu'il faut construire les requêtes : les propriétés recherchées pour chaque entité peuvent être spécifiques à une classe, mais pas à l'entité elle-même. En suivant la classification de *Wikidata*, les entités présentes dans le corpus appartiennent aux catégories suivantes :

— personnes humaines ; cette catégorie est la plus fréquente (12090 occurrences)

— noms de familles (3180 entités)

— communes françaises (586 occurrences)

— peintures et sculptures (respectivement 520 et 236 entités)

Cette variété – présentée sous forme graphique en annexes (2) – peut s'expliquer en partie par le taux d'erreur dans l'alignement avec *Wikidata* : il est par exemple probable que les peintures et sculptures soient sur-représentées parmi les entités *Wikidata* liées

aux catalogues. Cependant, toutes ces « erreurs » ne correspondent pas forcément à des résultats qui ne sont pas pertinents. Par exemple, l'algorithme peut aligner un.e écrivain.e avec un de ses ouvrages, ou une personne avec son portrait. Des résultats erronés peuvent toujours garder une forme de pertinence. Il est d'autant plus important de construire des requêtes `SPARQL` qui se concentrent pas uniquement sur des personnes. Cependant, il n'est pas possible de s'adapter à l'intégralité de la diversité du corpus. Le choix a donc été fait de se concentrer sur les catégories les plus pertinentes : les personnes, les familles, et les œuvres artistiques et littéraires. Non seulement ces catégories contiennent la grande majorité du corpus (16026 entités, soit 84,79%), mais ces catégories sont les plus à même de contenir des entités pertinentes. Il a été choisi de ne pas faire de requête spécifique sur les lieux, puisque *Wikidata* présente peu d'informations pour les entités de la catégorie « communes françaises ».

Un nombre assez conséquent de données ont donc été requêtées avec `SPARQL`, du fait des spécificités des bases de données en graphes, de la variété des entités *Wikidata* auxquelles les manuscrits sont liées, et enfin du fait de la variété du corpus lui même. Ces informations récupérées correspondent aux différentes catégories de *Wikidata*.

— Pour les personnes et les familles, les informations suivantes sont récupérées sur *Wikidata* :
  — Le genre de la personne.
  — Sa nationalité, afin de voir si l'origine d'une personne influence le prix d'un manuscrit.
  — Les langues parlées par une personne ; là encore, l'objectif est d'étudier l'impact de l'origine d'un.e auteur.ice sur un prix.
  — Les date de naissance et de décès, afin de placer un manuscrit dans une époque et de voir comment son ancienneté ou sa contemporanéité en influencent le prix.
  — Le lieu où une personne est née, où elle a vécu et où elle est morte, pour des raisons analogues.
  — La manière dont la personne est morte. Si cette information peut sembler anecdotique à un public contemporain, les catalogues de ventes sont marqués par un goût du sensationnel, et la manière dont une personne est morte est souvent mentionnée, notamment en cas d'exécutions.
  — La religion d'une personne : il peut être intéressant d'étudier si, et comment, ce critère influence l'évolution d'un prix.
  — Les titres de noblesse d'une personne.
  — L'éducation qu'a reçu une personne, afin de mieux situer ses occupations et d'analyser l'impact du niveau et du type d'éducation sur le prix.

  - L'occupation d'une personne, et les fonctions précises qu'elle a occupé : là encore, il est intéressant de situer l'impact de la carrière sur le prix et de voir quelles occupations sont corrélées avec des prix élevés sur le marché des manuscrits.
  - Les prix et distinctions reçus par une personne. À l'aide de ce critère, il est alors possible de chercher à répondre à cette question : la célébrité d'une personne de son vivant impacte-elle le prix de se manuscrits ?
  - Les organisations et institutions dont la personne est membre (Académie française, Franc-maçonnerie...).
  - Le nombre d'œuvres écrites ou réalisées par une personne. Là encore, c'est une tentative de mesurer l'impact ou la célébrité de celle-ci : les manuscrits de quelqu'un ayant beaucoup écrit sont ils plus chers que les manuscrits d'une personne ayant peu écrit ?
  - Le nombre de conflits auxquels une personne a participé. Ce critère de recherche permet de quantifier l'importance d'un personnage militaire.
  - Des images, telles que le portrait et la signature.
- Pour les créations littéraires, ce sont des informations bibliographiques qui sont avant tout récupérées ; pour les autres œuvres d'art, des informations analogues sur le contexte de création sont retenues.
  - Le titre de l'œuvre.
  - Son auteur.ice, pour étudier si certain.e.s auteur.ice.s sont susceptibles d'influencer le prix d'un manuscrit.
  - La date de création de l'œuvre, afin de savoir si l'époque d'origine influence le prix. Pour les livres, la date publication est également récupérée.
  - La requête récupère aussi la maison d'édition d'un livre.
  - Les dimensions et matériaux d'une œuvre d'art sont également d'intérêt.
  - Enfin, le genre et le mouvement dans lequel s'inscrivent une œuvre sont d'intérêt : ces informations pourraient permettre de voir si une hiérarchie des du goût influence le prix d'un manuscrit.
- Pour finir, afin de pouvoir éventuellement enrichir nos données avec d'autres sources externes à *Wikidata*, des identifiants uniques ont été récupérés afin de donner accès à d'autres bases de données en ligne : les identifiants VIAF (Fichier d'autorité international virtuel), ISNI (*International Standard Name Identifier*), de la Bibliothèque nationale de France, de la Bibliothèque du Congrès américain, ainsi que les identifiants *IDRef*. Certaines insitutions, comme la BnF, rendent leurs données accessibles via `SPARQL` ; la récupération de ces identifiants faciliterait grandement les enrichissements ultérieurs depuis d'autres sources de données.

Comme on l'a dit, l'objectif principal de l'alignement avec *Wikidata* est de produire des données pour calculer des régressions linéaires, ce qui permettrait d'étudier les déterminants du prix d'un manuscrit sur le marché du XIX$^{\text{ème}}$ s.. Cette récupération d'informations en masse ouvre d'autres possibilités. Entre autres, de nombreuses données géographiques ont sont récupérées (lieu de naissance, de décès, d'inhumation et de résidence). Il serait ensuite possible de récupérer les coordonnées de ces lieux, afin de construire une cartographie des auteur.ice.s dont les manuscrits circulent sur le marché du XIX$^{\text{ème}}$ s. parisien. C'est d'ailleurs souvent à des visées cartographiques et de géoréférencement que sont menées des campagnes de reconnaissance d'entités nommées ; dans les dernières années en France, de nombreuses études ont été menées pour développer des cartographies à partir de textes littéraires français encodés en `TEI` (Soudani *et al.*[3, 4]). En développant une approche cartographique, le projet *MSS / Katabase* pourrait apporter de nouvelles possibilités pour de telles études : le corpus de catalogues étant une source secondaire sur l'histoire littéraire française, une approche géographique permettrait d'étudier les origines géographiques des auteur.ice.s, plutôt que d'analyser les lieux représentés dans leurs œuvres. Cette possibilité n'est pas anodine, puisqu'elle permettrait de mettre en relation la « parisianité » avec la construction du canon littéraire à Paris. Il serait également possible d'étudier la circulation des productions culturelles, et leur rayon d'influence. En croisant les données géoréférencées avec des informations chronologiques (dates de naissance et de décès...), ces questions peuvent également être étudiées de façon historique : comment l'influence de l'origine géographique sur la réception d'une œuvre évolue au fil des siècles ? Répondre à ces questions n'a pas été possible dans le cadre de mon stage ; cependant, grâce à l'enrichissement de données via `SPARQL`, de telles études deviennent possibles, et les données pour mener ces analyses sont au moins en partie déjà disponibles. Produire des informations normalisées et exploitables pour la recherche implique donc de produire des données qui puissent être réutilisées dans d'autres problématiques de recherches.

6.3 Présentation générale

L'algorithme de récupération des informations sur *Wikidata* est considérablement plus simple que ce qui a été présenté lors de l'extraction d'informations des catalogues et de l'alignement avec des entités *Wikidata* : lors de ces étapes, les principales difficultés résultaient du bruit dans les données et de la nature « semi-structurée » des entrées de

3. Id., « Adaptation et évaluation de systèmes de reconnaissance et de résolution des entités nommées pour le cas de textes littéraires français du 19ème siècle »…, p.4-5.

4. Francesca Frontini, C. Brando, Marine Riguet, Clémence Jacquot et Vincent Jolivet, « Annotation of Toponyms in TEI Digital Literary Editions and Linking to the Web of Data », *Matlit Revista do Programa de Doutoramento em Materialidades da Literatura*, 4–2 (11 juil. 2016), p. 49-75, DOI : 10.14195/2182-8830_4-2_3, p. 63-66.

catalogues, qui demandait de s'adapter à de nombreux cas de figure. Une fois que des identifiants *Wikidata* ont été extraits, le processus est bien plus simple – comme cela apparaît dans la figure 6.2. Un identifiant étant unique et servant de clé d'entrée à une base de données en graphe très structurée, un comportement uniforme peut être adopté pour récupérer des données issues de *Wikidata*. Il ne s'agit plus que, pour chaque identifiant *Wikidata* récupéré, de lancer des requêtes `SPARQL` et d'en stocker le résultat dans un fichier. Le comportement de l'algorithme est donc toujours le même.

L'algorithme commence par vérifier si un identifiant a déjà été traité dans un fichier de log ; l'utilisation de ces fichiers évite d'avoir à recommencer la récupération d'informations de *Wikidata* à chaque interruption du script, ce qui est essentiel puisque cette étape dure plus de dix heures. Si l'identifiant n'a pas été traité, alors plusieurs requêtes `SPARQL` sont lancées sur *Wikidata*. Les résultats sont retournés en `JSON` ou `XML` dans des formats définis dans une spécification du W3C [5]. Ceux-ci ont un but descriptif (ils décrivent la requête et les données retournées) ; les documents `SPARQL` sont donc très complets et peu malléables ; c'est pourquoi ils sont transformés en une base de connaissances au format `JSON`. Celle-ci, comme nous le verrons, ne contient que les données obtenues dans un format plus malléable.

Les requêtes `SPARQL` sont faites sur un serveur distant via le protocole HTTP et demandent parfois au serveur de traiter de très grandes quantités de données. Des erreurs peuvent donc avoir lieu. Pour permettre au script de fonctionner malgré ces erreurs, un système de gestion des erreurs a donc été mis en place. Par défaut, il est demandé au serveur de *Wikidata* de fournir les résultats de la requête en `JSON`, format très léger et malléable. Deux erreurs peuvent alors arriver : soit le `JSON` retourné par le serveur est mal formé (et ne peut donc être traité), soit la requête excède la durée maximale autorisée (qui est d'une minute). Dans ces cas, la requête est lancée une seconde fois au serveur, mais cette fois-ci le format de réponse demandé est le `XML` ; cela permet d'éviter que le document soit mal formé ; il est également possible que la requête s'exécute dans le temps imparti la seconde fois. Si une erreur a encore lieu à la seconde requête, alors une entrée vide est associée à l'identifiant *Wikidata* dans la base de connaissances qui est en train d'être constituée. Cela signifie qu'aucune réponse n'a pu être obtenue.

5. Dave Beckett, Jeen Broekstra et Sandro Hawke, *SPARQL Query Results XML Format (Second Edition). W3C Recommendation 21 March 2013*, W3C, 21 mars 2013, URL : `https://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/` (visité le 10/05/2022).

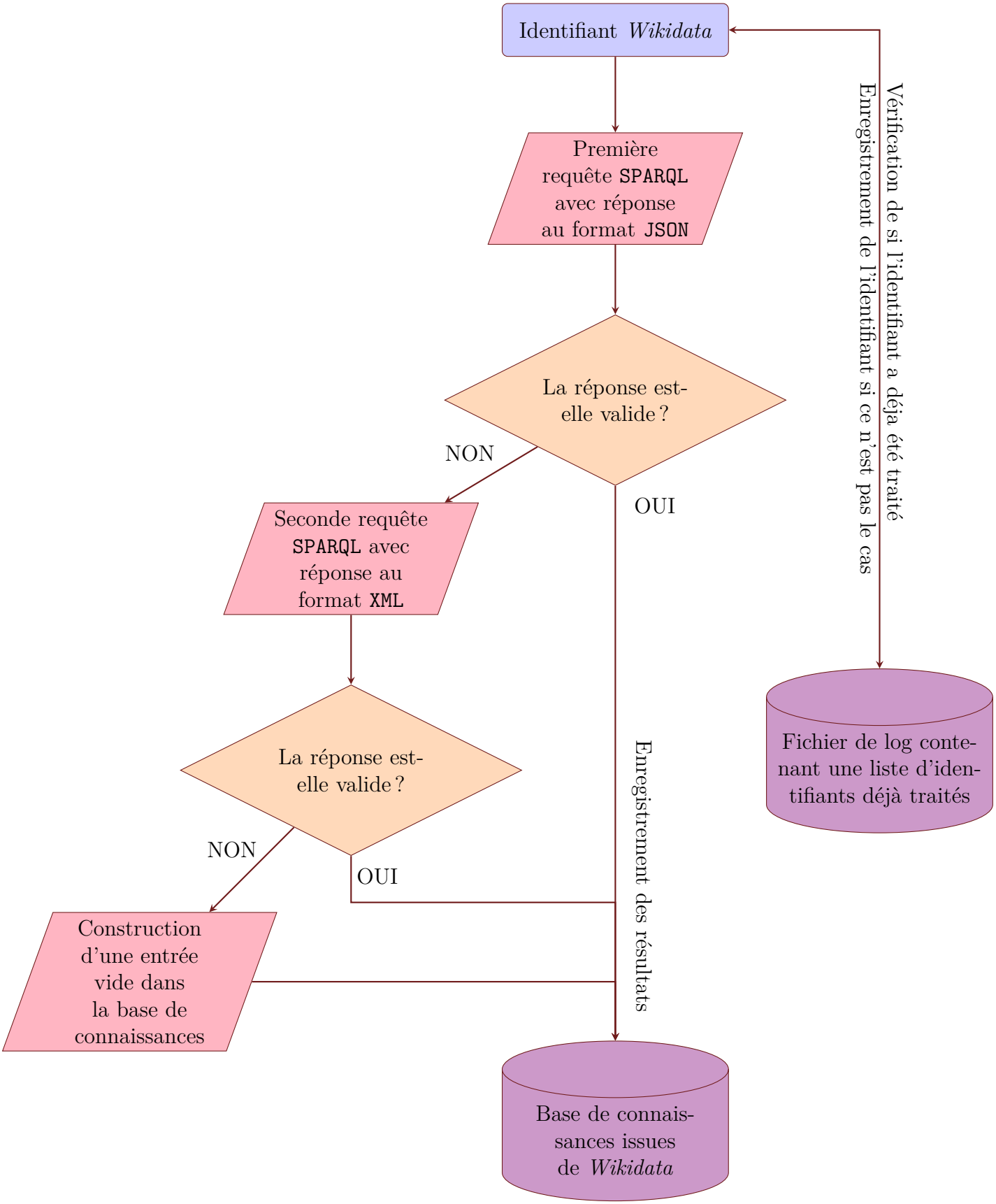


FIGURE 6.2 – L'algorithme de constitution d'une base de connaissances issues de *Wikidata* à l'aide de SPARQL

6.4 Développer un comportement uniforme pour produire des données exploitables à partir un corpus hétérogène

6.4.1 Les spécificités techniques de SPARQL : adapter les résultats à ses besoins

Le langage de requêtes

D'un point de vue technique, SPARQL suit la logique des « triplets » présentée plus haut. Récupérer des données via SPARQL revient à demander au langage d'extraire de la base de données l'ensemble des valeurs qui correspondent à un triplet sujet-prédicat-objet. Par exemple, récupérer tous les tableaux peints par Natalia Gontcharova est possible à l'aide de la requête suivante (6.1) :

```
SELECT ?painting ?paintingLabel
WHERE {
    ?painting wdt:P170 wd:Q232391 .

    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "en" .
    }
}
```

Code source 6.1 – Une requête simple : les identifiants et les noms de tableaux peints par Natalia Gontcharova

La requête ci-dessus consiste à récupérer tous les résultats possibles pour la variable `?painting` qui ont la relation `P170` (« a pour créateur.ice ») avec l'entité `Q232391` (l'identifiant de N. Gontcharova) : la requête récupère toutes les entités de *Wikidata* ayant l'artiste pour créatrice. La partie suivant `SERVICE...` permet d'associer à chaque `?painting` une variable `?paintingLabel` qui contient le label de l'entité `?painting` en anglais. Chaque entité ou relation dispose d'un préfixe (`wdt:, wd:, wikibase:`) qui permet de lier une information à un espace de noms spécifique. De cette brève description du fonctionnement technique de SPARQL, il faut retenir que ce langage permet de récupérer d'une base de données toutes les informations qui correspondent à la proposition énoncée. Celles-ci sont liées à des variables, ce qui permet d'accéder aux résultats renvoyés par SPARQL.

À l'origine, toutes les informations étaient demandées à SPARQL dans une seule requête. Celle-ci fonctionnait sans difficultés pour certaines entités ; dans d'autres cas cependant, elle ne s'exécutait pas dans le temps imparti. Plusieurs tentatives d'optimi-

sation ont été menées, mais sans succès. Ce cas de figure, comme cela est noté [6] par la documentation SPARQL de *Wikidata*, est systématique lorsqu'un très grand nombre d'informations sont récupérées pour une variable requêtée. Cela fait exploser la complexité du traitement de la requête par le serveur et cause un dépassement du temps maximal de requête autorisé. Le problème, c'est qu'il est difficile de prévoir les cas de figure où trop de résultats seront récupérés par SPARQL. Le choix a donc été d'adopter un comportement uniforme pour tous les identifiants. Plutôt que d'adapter les requêtes en fonction du type d'entités requêtées, la requête est subdivisée en quatre ; leurs résultats sont agglomérés afin qu'une entité soit associée à une seule série de résultats. Les risques de dépassement du temps maximal de traitement autorisé par *Wikidata* ont été pris en compte : si cette durée est dépassée, la requête est relancée ; si aucune réponse n'est obtenue cette seconde fois, alors une série de résultats vides sont associés à l'entité problématique dans la base de connaissances.

Le format de réponse

La réponse retournée par *Wikidata* ne peut être utilisée en tant que telle et doit donc être retravaillée. En effet, en plus de contenir les données renvoyées par SPARQL, les formats de réponse définis par le W3C [7] décrivent la requête (en stockant un ensemble de variables) et les données retournées (en définissant systématiquement un type de données, et en associant à ce type une URI). Comme cela a été dit, une requête SPARQL associe une variable à une « question » exprimée sous la forme d'une relation sujet-prédicat-objet ; celle-ci permet d'associer une ou plusieurs valeurs à la variable requêtée. Une requête comprenant plusieurs variables peut donc retourner un nombre de valeurs différent pour chacune de ces variables. SPARQL ne peut alors pas déterminer de lien logique entre les valeurs obtenues pour ces variables. Le langage de requêtes calcule alors le produit cartésien entre toutes les valeurs possibles pour chaque variable : il y a alors autant de « solutions à la requête » [8] qu'il y a de combinaisons possibles de résultats. Par exemple, si trois variables sont requêtées, que SPARQL retourne trois résultats pour chaque variable et qu'il n'existe aucun lien logique entre elles (la valeur d'une variable ne dépend pas de la valeur des autres), alors le nombre de « solutions à la requête » est de $3 \times 3 \times 3 = 27$. Des exemples de réponse aux formats JSON (13) et XML (14) sont disponibles en annexes. Dans les deux cas, la structure est la même :

— Un en-tête (`head`) contenant une liste des variables requêtées.

— Un corps de texte (`results`) contenant une liste de toutes les « solutions de requêtes possibles » (`bindings`), c'est-à-dire toutes les combinaisons des valeurs obte-

6. *Query optimization*, Wikidata, 2022, URL : https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/query_optimization (visité le 06/06/2022).

7. Id., *SPARQL Query Results XML Format (Second Edition)*…

8. *Ibid.*, §2.3.1. Variable Binding Results.

nues pour les variables requêtées. Chacune de ces solutions contient une entrée par variable obtenue. Pour chaque variable, plusieurs informations sont retournées :

— La valeur obtenue

— Le type de cette valeur, qui permet de classifier la valeur en grands groupes : une URI, du texte libre...

— Si le résultat obtenu est en texte libre, alors la langue de ce texte peut également être fournie.

Cette variété d'informations documente précisément les données associées à chaque variables ; cependant, cette structure imbriquée rend l'accès aux informations très difficile. Par conséquent, le format de réponse de `SPARQL` présente deux problèmes. D'abord, il contient des informations inutiles pour la base de connaissances qui doit être constituée : celle-ci n'a pas besoin du le langage ou du type des valeurs récupérées. Ces informations étant définies dans la requête, leur valeur est déjà connue et peut être documentée sans alourdir la base de connaissances à construire. Ensuite, une réponse `SPARQL` peut potentiellement contenir énormément de redondances. Dans les faits, il n'est pas nécessaire de déterminer un lien entre les différentes variables requêtées, puisque celles-ci sont indépendantes. Ce qui importe, ce n'est pas le lien entre les distinctions obtenues par une personne et les positions qu'il ou elle a occupé (cette relation n'existe pas), mais le lien entre l'entité *Wikidata* sur qui est faite la requête et les valeurs obtenues : ce qui est nécessaire, c'est de savoir que telle valeur obtenue entretient telle relation avec l'entité requêtée (par exemple, de savoir que « Membre de l'Académie Française » est une distinction honorifique).

6.4.2 La base de connaissances à construire

L'objectif de cette base de connaissances est de stocker toutes les informations obtenues avec `SPARQL` pour toutes les entités, afin de faire le lien entre les auteur.ice.s de manuscrits et une source de données externe. Par conséquent, cette base de connaissances doit remplir les conditions suivantes :

— Permettre un accès facile aux données : cette base n'est pas censée être accessible au public, mais être facilement manipulable par une machine.

— Être construite dans un format léger et manipulable. Là encore, la base n'est pas censée être qualitative scientifiquement (contrairement à une édition textuelle en `TEI`), mais être utilisable rapidement par une machine. Le format utilisé doit donc favoriser le traitement des données, et non leur publication.

— Limiter au maximum les redondances d'informations : les requêtes contenant plus de 18000 identifiants, la base constituée sera probablement volumineuse ; plus elle occupe de mémoire, plus il est coûteux (en temps et en charge processeur) d'y récupérer des données.

— Être malléable : le format et le schéma définis pour la base de données doit permettre de contenir, pour chaque variable, un nombre indéfini de valeurs.

— Être accessible depuis les identifiants des entités *Wikidata* : c'est eux qui permettent le lien avec les catalogues.

Trois formats ont été considérés pour la base de connaissances :

— Le tableur (type `TSV`) : connu et utilisé par tout.e chercheur.euse, il a l'avantage d'être très facilement compréhensible et consultable par un public non-spécialiste. Dans ce cas, chaque colonne correspondrait à une variable et chaque ligne à une entité, ce qui permet à la fois un accès rapide aux identifiants et aux résultats de requête.

— Une base de données document au format `XML`. Ce langage à balises a l'avantage d'être hiérarchisé et de permettre des arborescences très complexes. Il est de plus possible de s'aligner sur des standards, comme le `XML-TEI` ; de plus, il est possible de concevoir des schémas de validation pour un fichier `XML` ; ceux-ci garantissent qu'il n'y ait pas d'erreur dans la structure de données, ce qui peut être intéressant puisque la base de connaissances est construite de façon automatique, et qu'une erreur est donc potentiellement possible.

— Une base de données `JSON`. Ce format, également hiérarchisé, permet de construire des structures arborescentes (presque) aussi complexes que le `XML`. Il est par contre bien plus léger et facile d'utilisation que ce dernier ; contrairement aux langages à balises, le balisage sémantique est moins facile (il est moins évident d'associer une signification à chaque élément). De plus, il n'est à ma connaissance pas possible de construire des schémas de validation pour un `JSON`.

Étant donné qu'il est impossible de prévoir le nombre de valeurs obtenues pour une variable à chaque requête, un format non-hiérarchisé est peu intéressant : dans un tableur, il n'est possible de stocker qu'une information par colonne/variable, à moins de créer des subdivisions au sein d'une colonne, ce qui rendrait le tableur plus complexe à manipuler. Au contraire, un format hiérarchisé permet d'associer chaque variable à une liste de résultats dont la longueur n'est pas définie par avance. Le `TSV` a donc été disqualifié pour l'accueil des données. Le `JSON` a enfin été privilégié au `XML` : ce dernier est considérablement plus complexe à manipuler, que ce soit pour y ajouter des informations que pour accéder à celles-ci. De plus, le langage à balises est bien plus lourd, et donc potentiellement plus long à manipuler. Étant donné que la base de connaissances est un format de « travail », et non de publication, un balisage sémantique n'est pas nécessairement intéressant. C'est donc le `JSON` qui correspond le mieux aux besoins du projet.

La structure suivante a été définie pour la base de connaissances : à chaque identifiant *Wikidata* est lié les résultats obtenus via `SPARQL`. Ces résultats correspondent à toutes les variables requêtées ; à celles-ci sont associées une liste de résultats. Toutes les variables

requêtées sont toujours présentes. Si aucun résultat n'a été trouvé pour une variable, alors celle-ci est associée à une liste vide. Cela permet que toutes les entrées de la base aient toujours la même structure. Ainsi, la manipulation de la base est facilitée ; en contrepartie, celle-ci est plus lourde que si les seules variables présentes étaient celles pour lesquelles des résultats ont été trouvés. En d'autres termes, la base de connaissances correspond au schéma ci-dessous (6.2). Un exemple contenant deux entrées de la base figure en annexes (15).

```
{
    "identifiant wikidata 1": {
        "variable 1": ["résultat 1", "résultat 2", "résultat N"],
        "variable 2": ["résultat 1", "résultat 2", "résultat N"],
        "variable N": ["résultat 1", "résultat 2", "résultat N"],
    },
    "identifiant wikidata N": {
        "variable 1": ["résultat 1", "résultat 2", "résultat N"],
        "variable 2": ["résultat 1", "résultat 2", "résultat N"],
        "variable N": ["résultat 1", "résultat 2", "résultat N"],
    },
    # autres entités...
}
```

Code source 6.2 – Structure de la base de connaissances construite à partir des résultats obtenus de `SPARQL`

6.4.3 Des réponses `SPARQL` à la base de connaissances

Pour produire des données utilisables dans un contexte de recherche, il est nécessaire de retraiter le format de réponse de `SPARQL` pour s'adapter au modèle de données défini ci-dessus. Cependant, le contenu d'une réponse `SPARQL` varie selon la requête lancée, selon l'identifiant *Wikidata* requêté et enfin selon les informations retournées par cette base de connaissances. En effet, une variable requêtée n'apparaît dans le corps des résultats que si des valeurs correspondant à cette variable se trouvent dans *Wikidata*. Il faut donc trouver une méthode uniforme pour traiter toutes les réponses de `SPARQL`, peu importe la requête lancée ou les résultats obtenus. C'est dans l'en-tête des réponses que se trouve la réponse à ce problème : celui-ci contient l'ensemble des variables requêtées, que des résultats y soient associés ou non (voir la clé `vars` dans le `head` d'une réponse `SPARQL` : annexe 13). Pour construire la base de connaissances à partir des réponses `SPARQL`, il suffit donc de récupérer ces variables dans l'en-tête, et ensuite d'extraire les valeurs qui y sont associées dans les différentes « solutions de requêtes » contenues dans le document. Il est cependant nécessaire de supprimer les redondances dues au calcul d'un produit cartésien

entre les résultats. Pour ce faire, un valeur n'est liée à un résultat que si elle n'a pas déjà été extraite.

Une difficulté supplémentaire apparaît si une erreur a lieu au lancement d'une requête. Dans ce cas, la requête est relancée, mais cette fois-ci, le format de réponse demandé est le `XML`, au lieu du `JSON`. Cela permet d'éviter d'éventuels problèmes dans la construction de la réponse `JSON` ; relancer la requête permet aussi de faire une seconde tentative lorsque la première requête a dépassé la durée maximale d'exécution autorisée. Le script de traitement des réponses décrit ci-dessus ne fonctionne cependant pas avec des données en `XML`. Plutôt que de réécrire l'intégralité de ce script pour du `XML` (ce qui serait également possible), le choix a plutôt été fait de traduire un réponse `XML` en un document `JSON` conforme avec la spécification `SPARQL`[9]. Ce `JSON` a une structure qui permet la traduction des résultats `SPARQL` pour constituer la base de connaissances. Le processus de transformation du `XML` en `JSON` en lui-même est assez simple, puisque les deux formats de réponse ont la même structure, mais simplement une syntaxe formelle différente. Ainsi, plutôt que de traduire directement la réponse `SPARQL XML` obtenue au format voulu pour la base de connaissances, il est plus facile de passer par une transformation intermédiaire en `JSON`. Une fois ce traitement réalisé pour les quatre requêtes faites par entité, il ne reste plus qu'à ajouter à la base de connaissances ces données récupérées de *Wikidata*.

9. *Ibid.*

Chapitre 7

Un corpus augmenté : enrichissement sémantique du corpus et possibilités ouvertes par l'alignement avec *Wikidata*

7.1 Lier la TEI aux données nouvellement produites

Jusqu'alors, les catalogues ne servent que de sources de données qui permettent de réaliser l'alignement : ils n'ont pas étés modifiés durant tout ce processus. Cependant, d'importantes modifications ont eu lieu pour l'ensemble du projet : les auteur.ice.s de manuscrits ont été liés à entités *Wikidata*, ce qui a permis la constitution d'une base de données « sur mesure ». Celle-ci contient des informations pertinentes pour mener une étude économétrique qui cherche à faire le lien entre des facteurs biographiques et le prix d'un manuscrit ; à l'aide de ces données, il est possible de mieux évaluer la célébrité d'un.e auteur.ice, voire peut-être de comprendre qu'est-ce qui fait cette célébrité. Comme nous allons le voir plus bas, cette base de données peut être utilisée au delà de cette étude économétrique.

La première étape, cependant, est d'expliciter la relation entre les catalogues et cette source de données externe. Il est donc nécessaire d'annoter le corpus de catalogues à l'aide des identifiants *Wikidata* : il s'agit d'un enrichissement sémantique, qui permet, pour une entrée de catalogue, d'accéder à une entrée de la base de connaissances sur les auteur.ice.s. Grâce à son usage des attributs, la TEI rend possible l'annotation de corpus de façon assez intéressante. Un attribut permet de rajouter ou de normaliser des informations sur un élément sans en modifier le contenu textuel. Là où le corps du texte est situé entre une balise ouvrante et une fermante, un attribut se trouve dans la balise ouvrante. Les attributs permettent, entres autres, des liens entre différentes parties du document, ou,

ce qui nous intéresse dans ce contexte, avec des données extérieures. Il existe plusieurs manières, plus ou moins complexes, de créer des relations entre des éléments d'un texte et des ressources externes avec la TEI. En suivant les préconisations des *Guidelines*[1], les identifiants *Wikidata* ont été inclus dans un attribut `@ref`. Cet attribut permet de pointer vers une ressource externe.

Pour tirer véritablement parti d'un encodage numérique, l'alignement entre entités *Wikidata* et auteur.ice.s de manuscrits va un peu plus loin. Dans les catalogues mis à jour, ces identifiants sont précédés du préfixe `wd` : `<name type="author" ref="wd:Q231457">Marie-Amélie</name>`. Celui-ci est une abréviation de *Wikidata*. Il permet à un.e lecteur.ice humain.e de comprendre que l'attribut `@ref` renvoie à cette base de connaissances ; mais il permet surtout de créer des liens dynamiques vers celle-ci. En effet, ce préfixe est déclaré dans l'en-tête des catalogues (le `tei:teiHeader`) à l'intérieur d'un élément `tei:prefixDef` (visible en annexes : 16). Là, il est indiqué pour les lecteur.ice.s que le préfixe renvoie à *Wikidata*. Mais cet élément est surtout intéressant lors du traitement du document par une machine. Il y est indiqué que ce préfixe peut être remplacé par l'URL `https://www.wikidata.org/wiki/`. Ainsi, un processeur peut remplacer tous les préfixes par des URL, et ainsi reconstruire des liens vers les pages des entités sur *Wikidata*. Cette manière de lier un `tei:name` à un identifiant *Wikidata* permet d'expliciter les liens entre les catalogues et la base de données propre au projet constituée à l'aide de `SPARQL` ; elle permet également de faire des renvois dynamiques vers des ressources en ligne, facilitant l'accès d'utilisateur.ice.s à celles-ci. Cette double méthode de renvoi permet à la fois un meilleur traitement par une machine et une lecture « augmentée » grâce à des liens hypertextes. La transformation des identifiants en URL complets peut également être explicitement mise en place dans des étapes de traitement ultérieures des catalogues. Il serait par exemple possible d'intégrer ces hyperliens au site Web de *Katabase*. Ainsi, le traitement automatisé des catalogues et l'alignement avec *Wikidata* va dans la continuité des objectifs de la TEI : permettre la création de documents enrichis pour des lecteur.ice.s et manipulables par des machines.

7.2 Que faire des entités ?

Le processus décrit dans cette partie résolution d'entités nommées à l'aide de *Wikidata* et de construction d'une base de connaissances a été commencé avec une visée précise. Récupérer une grande quantité de données biographiques sur les auteur.ice permettrait de comprendre ce qui fait le prix d'un manuscrit. L'alignement avec *Wikidata* et la récupération de données via `SPARQL` ayant pris un temps important, cette étude n'a

1. TEI Consortium, *P5 : Guidelines for Electronic Text Encoding and Interchange*, Text Encoding Initiative, Version 4.4.0, 2022, URL : `https://tei-c.org/release/doc/tei-p5-doc/en/html/index.html`, 16. Linking, Segmentation and Alignment.

malheureusement pas pu être menée à son terme.

Au vu du temps passé à produire ces données, il serait cependant dommage de limiter l'usage des données obtenues à cette étude. En effet, la résolution d'entités nommées ouvre énormément de possibilités : elle permet d'obtenir des données structurées et manipulables par une machine, et donc de réaliser un traitement et une analyse « de masse » de documents textuels. Comme cela a déjà été dit, la résolution d'entités a souvent, en humanités numériques, des visées cartographiques. Cette cartographie est souvent – et logiquement – consacrée à des lieux physiques [2]. Le *Bentham Project* a proposé une approche intéressante : une cartographie d'un corpus du philosophe Jeremy Bentham (philosophe utilitariste célèbre pour ses travaux sur le panoptique) encodé en `XML-TEI`. Les entités traitées par le projet Bentham dépassent les personnes, puisque des concepts sont extraits et liés à *Wikidata* ; elles sont identifiées par comparaison à des concepts présents sur la base en ligne *DBPedia* et en extrayant les thèmes principaux du corpus [3]. L'application Web conçue pour le corpus tire partie de ce liage d'entités : les concepts sont représentés dans des réseaux interactifs navigables ; parmi ceux-ci, les plus importants dans la production de J. Bentham pour chaque décennie sont visibles sous la forme de cartes de fréquentation (*heatmaps*) [4]. Ce projet – fruit d'une campagne de transcription participative (*crowdsourcing*) [5] sur plus de huit ans – est d'une ampleur sans commune mesure à celle de *MSS / Katabase*. Le projet offre tout de même des pistes intéressantes pour le développement Web dans un projet d'humanités numériques (ce qui constitue la troisième partie du présent mémoire). Au delà de l'utilisation de données produites lors de cette étape en interne, à des fins de recherche, il serait intéressant de diffuser les données au public via l'application Web *Katabase*.

Au delà de questions cartographiques, la base de connaissances constituée ouvre la voie à d'autres analyses du corpus : elle contient des informations qui ne sont pas explicitement présentes dans le corpus de catalogues. L'extraction d'informations de *Wikidata* permettrait de procéder à la reconnaissance d'entités nommées dans les parties les moins structurées des catalogues (c'est-à-dire, les `tei:note` et les `tei:desc`, souvent riches d'informations sur les destinataires des lettres) : il serait possible, uniquement par détection de motifs, d'identifier les destinataires de lettres qui sont également auteur.ice.s de ma-

2. A. Soudani, Y. Meherzi, A. Bouhafs, *et al.*, « Adaptation et évaluation de systèmes de reconnaissance et de résolution des entités nommées pour le cas de textes littéraires français du 19ème siècle »… ; Francesca Frontini, C. Brando, M. Riguet, *et al.*, « Annotation of Toponyms in TEI Digital Literary Editions and Linking to the Web of Data »… ; Noémie Boeglin, Michel Depeyre, Thierry Joliveau et Yves-François Le Lay, « Pour une cartographie romanesque de Paris au XIXe siècle. Proposition méthodologique », dans *Conférence Spatial Analysis and GEOmatics*, Nice, France, 2016 (Actes de la conférence SAGEO'2016 - Spatial Analysis and GEOmatics), URL : `https://hal.archives-ouvertes.fr/hal-01619600`.

3. Pablo Ruiz Fabo et Thierry Poibeau, « Mapping the Bentham Corpus : Concept-based Navigation », *Journal of Data Mining & Digital Humanities*, Atelier Digit_Hum (Data deluge : which skills for...[2019]), p. 5044, DOI : `10.46298/jdmdh.5044`, p. 8-11.

4. *Ibid.*, p.14-17.

5. *Ibid.*, p. 2.

nuscrits présents dans le corpus. Plutôt que d'utiliser l'apprentissage machine (pour lequel des méthodes sont déjà mises en place et pratiquées), coûteux en ressources, la présence de données structurées pourrait de continuer à développer des approches de « basse technologie » pour analyser un corpus semi-structuré. Grâce à cette base, il devient par également possible d'étudier la répartition des genres des auteur.ice.s de manuscrits : leur genre a été récupéré dans des requêtes `SPARQL`. Il a été envisagé de mener une telle étude au début de mon stage. Il est cependant difficile de faire cette analyse en s'appuyant uniquement sur les catalogues : le genre n'y est pas explicitement donné. Il est difficile d'appliquer des méthodes de détection de motifs à la question du genre d'un.e l'auteur.ice. Il serait possible de détecter les adjectifs au féminin dans le `tei:trait` ; cependant, de telles informations manquent souvent des entrées de catalogue. Une identification du genre à partir des prénoms serait presque irréalisable, du fait de la variété des prénoms, de la présence de prénoms non genrés dans la langue française, ou encore de l'absence de règles précises distinguant un prénom masculin et féminin. La résolution d'entités avec *Wikidata* permet cependant de mener de telles études. Le genre d'un.e auteur.ice pourrait être annoté dans les corpus, comme cela a été fait pour les identifiants *Wikidata*. Étant donné que les entités permettent d'expliciter le genre, elles permettent également la constitution d'une base de connaissances contenant des noms et prénoms masculins et féminins ; il serait alors possible d'identifier la présence de ces prénoms dans le texte afin d'annoter automatiquement des corpus de catalogue avec ces informations.

Enfin, l'alignement des auteur.ice.s de manuscrits peut permettre des enrichissements ultérieurs : plusieurs identifiants ont été récupérés en plus de l'identifiant *Wikidata* (« VIAF, DataBnF »...). Cela facilite les enrichissements ultérieurs de la base de connaissances constituée lors de cette étape en réalisant d'autres requêtes. Il serait également possible d'enrichir les contenus diffusés au public via l'application Web *Katabase*, en créant par exemple des pages pour les auteur.ice.s. Jusqu'à maintenant, cela n'a pas été fait car les manuscrits sont pauvres en informations biographiques. Ce cas de figure (une grande quantité d'entrées contenant peu d'informations) est similaire à celui présenté par Aguirre *et al.*[6] au sujet des collections de *Digitised cultural heritage* (« Partimoine culturel numérisé »). Ces collections, composées de millions de cartels d'œuvres encodées en `XML` en suivant le standard « Dublin Core », contient très peu d'informations sur chaque item. Leur proposition est de développer une chaîne de traitement sur le modèle du liage d'entités, afin de lier les collections à des pages *Wikipedia*. Grâce à l'alignement avec *Wikidata* déjà réalisé dans le cadre du projet *Katabase*, il n'est pas nécessaire de mener une nouvelle campagne de résolution d'entités. À partir d'une simple requête `SPARQL`, les URL

6. Eneko Agirre, Ander Barrena, Oier Lopez de Lacalle, Aitor Soroa, Samuel Fernando et Mark Stevenson, « Matching Cultural Heritage items to Wikipedia », dans *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, dir. Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk et Stelios Piperidis, Istanbul, Turkey, 2012, p. 1.

des pages *Wikipedia* liées aux auteur.ice.s peuvent être automatiquement récupérées. Avec ces URL, il est facile d'automatiser la récupération du texte d'articles de *Wikipedia* pour enrichir le site Web de *Katabase*.

7.3 En conclusion

Le processus d'alignement avec *Wikidata* présenté dans cette partie propose de s'appuyer sur des méthodes de détection de motifs à des fins de NEL. Ce choix technique est simple, par opposition aux outils d'apprentissage machine généralement utilisées à cette fin. Grâce à la nature semi-structurée du corpus, il est cependant possible d'identifier très précisément les éléments pertinents pour l'interaction avec *Wikidata*. Il est également possible d'attribuer à ces éléments une valeur sémantique, ce qui permet de construire un algorithme de requêtes réactif. L'utilisation de technologies « simples » a cependant été contrebalancé par la conception d'algorithmes complexes, qui permettent de s'adapter de très près aux documents sources. La mise au point d'un algorithme construisant et faisant des requêtes sur le moteur de recherche de *Wikidata* contraste également avec les processus habituellement utilisés pour la résolution d'entités nommées. Dans le cas de `REDEN`, celle-ci s'appuie sur des graphes qui représentent les candidats possibles pour un alignement ; le candidat sélectionné pour la résolution est choisi en fonction de sa centralité dans le graphe [7] ; l'ensemble du processus repose sur de l'apprentissage machine. Cependant, les méthodes « traditionnelles » de NEL développent leurs propres méthodes d'alignement et de sélection des candidats ; celles-ci ne s'appuient pas sur l'utilisation de moteurs de recherche déjà existants et performants. Elles ne semblent pas non plus prendre en compte les informations biographiques qui ont été utilisées ici. À l'inverse, l'alignement par détection de motifs s'appuie fortement sur l'utilisation d'un moteur de recherche, et met donc à profit les outils déjà mis au point par les bases de connaissances. De plus, la sélection du candidat présentée ici s'appuie sur l'extraction d'un maximum d'informations biographiques de la source, qui ne sont pas utilisées en NEL. Dans une démarche d'alignement d'entités nommées, s'appuyer au maximum sur les données disponibles dans le texte source peut être une perspective intéressante. En maximisant la quantité d'informations extraites, il est possible de réduire le bruit et de sélectionner le bon candidat avec une plus grande certitude. Le principal problème des méthodes d'alignement avec *Wikidata* présentées ici est qu'elles sont difficilement portables sur d'autres données : les motifs repérés, les tables utilisées pour convertir des données et les algorithmes d'extraction ne fonctionnent qu'avec des catalogues de ventes de manuscrits. L'utilisation de l'API *Wikidata* est, elle, plus facilement adaptable. À l'inverse, l'apprentissage machine offre la promesse d'une plus grande portabilité ; il est cependant nécessaire d'entraîner des modèles afin de les adapter aux données voulues.

7. C. Brando, Francesca Frontini et J.G. Ganascia, « REDEN… », p. 72.

Il est intéressant de remarquer que les questions soulevées lors de cette étape se rapprochent des problématiques propres à l'apprentissage machine. Le premier problème apparent est un possible biais dans le processus d'extraction d'informations. Celui-ci est le fruit d'une analyse du corpus, d'une manipulation du moteur de recherche de *Wikidata* et de nombreux tests ; cependant, il est possible que certains aspects du corpus aient été par erreur favorisés. Ce problème se retrouve dans les données de test utilisées : puisque ces données ont fait l'objet de plus d'attention que le reste du corpus, il est possible que le processus d'extraction d'informations soit plus performant sur celles-ci que pour le reste du corpus. Cette difficulté rappelle le « surapprentissage » (*overfitting*), fréquent en apprentissage machine : un algorithme « apprend » trop efficacement des données sur lesquelles il est entraîné ; ses performances deviennent alors médiocres sur d'autres données, puisqu'il ne sait pas s'y adapter. Pour mesurer efficacement ce risque, deux jeux de données de test sont utilisés : le jeu d'entraînement (que l'algorithme utilise pour « apprendre ») et le jeu d'évaluation (qui mesure les performances). Ici, un seul jeu de données est utilisé, puisqu'il n'y a pas à proprement parler de phase d'entraînement. Il serait cependant utile de créer un jeu d'évaluation distinct afin d'évaluer les performances du liage d'entités. Cela permettrait de donner des résultats plus adéquats en évitant que l'algorithme soit évalué sur des données auxquelles il est trop adapté. En plus de ces problèmes liés à la source de données (ici, les catalogues), d'autres problèmes viennent de la base de connaissances utilisée dans la résolution d'entités. Comme le remarquent Carmen Brando *et al.*[8], la capacité à aligner une entité dans un texte avec la bonne entité sur une base de connaissances dépend largement des données présentes dans celle ci. L'extraction d'informations et l'utilisation du moteur de recherche de *Wikidata* ont cherché à contourner ce problème en traduisant les données sources en leurs équivalents sur *Wikidata*. Cela permet de se conformer aux biais de la base de connaissances, ce qui n'est pas possible avec des méthodes traditionnelles d'apprentissage machine : lorsqu'une une entité est résolue à partir de sa centralité dans un graphe, la différence de vocabulaires dans les sources et la base de connaissances ne peut pas être prise en compte.

Il va de soit que la détection de motifs n'est pas toujours possible : si elle fait sens pour un corpus semi-structuré, elle serait probablement moins efficace avec un texte littéraire, où les structures des phrases sont moins codifiées. L'encodage en `TEI` des sources textuelles peut, cependant, faciliter l'utilisation de telles méthodes sur des corpus plus complexes. En effet, puisqu'un tel encodage permet un balisage sémantique, il est possible de cibler des éléments précis à l'intérieur desquels détecter des motifs. Ainsi, il n'est pas nécessaire de trouver des méthodes qui fonctionnent pour l'intégralité du texte, et il est possible de travailler à une échelle restreinte.

Cette phase d'enrichissement à l'aide de données externes ouvre la voie à de nouvelles

8. C. Brando, N. Abadie et Fontini Frontini, « Évaluation de la qualité des sources du web de données pour la résolution d'entités nommées »…, p. 37-8.

études, et donc à une meilleure compréhension du corpus ; elle pourrait aussi permettre d'enrichir l'application *Katabase* avec de nouvelles fonctionnalités (des pages personnelles sur les auteur.ice.s de manuscrits, par exemple). Cependant, dans un projet d'humanités numériques, le Web ne doit pas uniquement être pensé comme un moyen de proposer du contenu « pour humain.e.s ». En effet, si un site Web traditionnel est facile à naviguer et pratique, il n'est pas toujours la meilleure manière de diffuser des informations. Cette partie s'est beaucoup attachée à démontrer l'intérêt d'utiliser des API dans le cadre d'un projet de recherche ; c'est pourquoi une telle application a été développée dans le cadre du projet *MSS / Katabase*. C'est cette API et ce qu'elle permet en termes de diffusion de données de recherche réutilisables, qui font l'objet de la prochaine partie.

Troisième partie

Rendre la recherche réutilisable et interopérable : *KatAPI*, une API pour échanger des données structurées

La problématique de cette partie est la suivante : comment rendre la recherche en humanités numériques réutilisable et encourager le partage de données entre projets de recherche ? La réponse à cette question, au sein du projet *MSS / Katabase* prend la forme d'une API. Celle-ci diffuse des données source du projet, mais aussi des informations issues de la recherche et du traitement computationnel de celles-ci dans deux formats structurés : le `JSON` et le `XML-TEI`.

En guise d'introduction, il me semble important ici de faire un bref rappel des principes fondamentaux du développement applicatif et de l'architecture du Web. Certains de ces principes ont déjà été présentés, mais ce rappel permettra de rendre la suite de ce chapitre plus claire.

Le Web est une architecture de type client-serveur qui permet l'interaction de machines dans un réseau décentralisé. Cette architecture définit deux fonctions possibles pour une machine : client (qui envoie et récupère des données distantes) et serveur (qui contient une application permettant l'interaction avec le client, ainsi qu'une base de données). Dans le Web, l'architecture client-serveur repose sur un protocole technique qui pose les principes de l'interaction entre des machines : le HTTP. Selon ce principe, l'accès d'un client à une ressource stockée sur un serveur (une page Web, par exemple) se fait au travers d'un URL. Celui-ci sert à localiser la ressource. L'interaction est définie par des requêtes (du client au serveur) et par des réponses (du serveur au client). Six types de requêtes sont possibles, qui correspondent à autant d'actions : demander des données à un serveur, en supprimer, les mettre à jour... Requêtes et réponses ont la même structure : elles sont composées d'un en-tête HTTP (qui contient des métadonnées sur la requête ou la réponse) et d'un corps (qui contient la requête ou la réponse en elle-même ; celui-ci peut être vide). Parmi les métadonnées contenues dans l'en-tête, deux sont importantes dans notre contexte (surtout pour les réponses du client au serveur) : le statut et le type MIME. Un statut est un code à trois chiffres qui caractérise la réponse et le déroulement de l'interaction client-serveur. « 200 » indique que tout va bien, « 404 » que la ressource demandée n'existe pas (c'est-à-dire, que l'URL de la requête ne pointe pas vers une ressource existante). Le type MIME est une manière standardisée de définir le format d'une réponse : « `application/json` » lorsque la réponse est au format `JSON`, « `application/xml` » lorsque du `XML` est renvoyé.

Une API est une application, c'est-à-dire un ensemble de fonctions rassemblées pour permettre l'interaction entre un client et un serveur. Souvent, et c'est le cas pour les applications du projet *MSS / Katabase*, une application permet l'interaction avec une base de données, elle-aussi stockée sur le serveur. Par exemple, lorsque l'on accède à une page *Wikipedia*, l'application reçoit une requête de la part d'un client ; elle va chercher dans la base les données pertinentes, les structure sous la forme d'une page HTML et les renvoie à l'utilisateur.ice, qui verra la page s'afficher sur son écran. La principale différence entre une API Web et une application (comme *Wikipedia*) est que l'API est « un site Web pour

machines ». Elle ne contient pas d'interface graphique et les réponses renvoyées par l'API sont des données dites « brutes » ; celles-ci ont l'avantage d'être lisibles et manipulables par une machine. L'interaction avec une API se fait en envoyant une requête à l'application avec un URL qui contient les paramètres de la requête et des valeurs associées à ces paramètres (quelles données demander, quel format de réponse utiliser...) et éventuellement un corps de requête qui contienne des données (ce qui n'est pas le cas lorsque la requête correspond à une demande de données). L'URL contient donc la sémantique de la requête, il correspond à une question posée à un serveur de façon structurée. L'intérêt d'une telle API est que, pour peu que l'on sache interagir avec elle, il est possible de récupérer automatiquement des données structurées, prêtes à être utilisées. Pour télécharger un jeu de données, il n'y a pas besoin, par exemple, d'aller sur la bonne page Web, de cliquer sur un bouton « Télécharger » et d'enregistrer le fichier au bon endroit. Toutes ces opérations peuvent être faites automatiquement, comme cela a été fait pour plus de 80000 requêtes lors de l'interaction avec *Wikidata* décrite précédemment. Cette perspective est particulièrement intéressante pour un projet de recherche, puisqu'elle peut permettre de récupérer automatiquement les jeux de données produits par un autre projet, ce qui encourage la réutilisation des données de la recherche. Pour des ingénieur.e.s des données, les API facilitent grandement le travail de récupération automatisée d'informations : elles permettent de « se concentrer sur le quoi, et non sur le comment » [9].

Souvent, un site Web peut être composé d'une version accessible pour des utilisateur.ice.s humain.e.s et pour des machines ; c'est le cas dans le projet *Katabase*. L'application pour humain.e.s a déjà été développée ; c'est la version pour les machines, son fonctionnement et son utilité pour permettre la réutilisation des données du projet qui est donc présentée dans cette partie.

9. *focus on the what, and not on the how.* Citation issue de Jaimie Murdock et Colin Allen, « InPhO for All : Why APIs Matter », *Journal of the Chicago Colloquium on Digital Humanities and Computer Scienc*, 1–3 (2011), URL : `http://www.jamram.net/docs/jdhcs11-paper.pdf` (visité le 12/07/2022), p. 2.

Chapitre 8

Standards de design et statut des API dans pour les humanités numériques centrées sur le texte

Il n'existe aucune norme communément acceptée pour la conception d'une API : dans l'absolu, tant que celle-ci fonctionne, elle peut être considérée comme une API. Cependant, des standards se sont développés afin de regrouper les différentes méthodes et de développer des bonnes pratiques de design. Deux types de standards sont ici présentés : les standards d'architecture et les standards d'interopérabilité. Un standard d'architecture ou de design, comme le REST, définit comment concevoir l'interaction client-serveur dans une API et proposent une manière de structurer les réponses du serveur. Il ne définit cependant pas quels paramètres de recherches sont autorisés, ni le corps d'une réponse doit être organisé. Il ne définit pas non plus précisément comment organiser une requête, c'est-à-dire quel URL construire pour accéder aux données. C'est le rôle des standards d'interopérabilité (*Open Archives Initiative Protocol for Metadata Harversting* (OAI-PMH) , *Canonical Text Services* (CTS) , *Distributed Text Services* (DTS)). Ceux-ci définissent la sémantique des requêtes (c'est-à-dire les paramètres autorisés) et la manière de construire un URL pour accéder à certaines données. Ils permettent donc l'interopérabilité, puisqu'une même syntaxe peut être utilisée pour requêter différentes API. Ni le REST, ni les standards d'interopérabilité ne définissent l'implémentation d'une API : les deux types de standards s'intéressent à l'interaction du client avec l'API, ils ne définissent pas l'application en elle-même. Une application qui se conforme à un standard garantit simplement un certain type d'interaction client-serveur. En plus de ces deux standards qui sont ici présentés, les principes *Findable Accessible Interoperable Reusable* (FAIR) sont également décrits : ceux-ci ne définissent ni implémentation, ni sémantique, mais sont simplement des principes généraux pour la manière dont des données doivent être partagées dans un contexte de science ouverte et réutilisable.

8.1 Que faire du FAIR ? Le partage des données de la recherche

Les principes FAIR [1] sont essentiels aux problématiques de la science ouverte. Celle-ci encourage non seulement la diffusion de la recherche dans des formats traditionnels (articles, conférences...), mais aussi la diffusion de données brutes issues de la recherche scientifique. L'objectif, dans la diffusion de ces données brutes, est de rendre les données réutilisables, mais aussi de permettre la reproductibilité de la recherche, et donc la vérification des résultats obtenus.

La problématique de la diffusion de données recherche brutes se retrouve bien sûr en humanités numériques. Dans le domaine de la reconnaissance optique de caractères, par exemple, se développe le partage de vérités de terrain. La reconnaissance de caractères est permise par l'entrainement d'un algorithme d'apprentissage machine sur des textes qui ont été transcrits manuellement ; c'est ce couple texte numérisé/transcription qui constitue la vérité de terrain. Or la production de ces données demande beaucoup de temps ainsi que des compétences spécifiques pour des graphies difficiles à lire et des langues rares [2]. Pour faciliter le développement de méthodes de reconnaissance optique de caractères se développent donc des initiatives pour faciliter le partage de vérités de terrains. Parmi ces initiatives se trouve le projet *HTR United*[3], qui offre un portail pour l'accès à et le dépôt de vérités de terrains. Celles-ci sont partagées avec des métadonnées précisant par exemple leur contexte de production, le domaine auquel appartient au corpus.

La diffusion de données de recherche est cependant une opération plus complexe qu'il ne pourrait y paraître ; c'est pourquoi les principes FAIR ont été édités en 2014 [4]. Ils forment une base minimale de recommandations sur la manière de partager des données de recherche. Cette base minimale est composée de quatre principes [5] :

— *Findability* : les jeux de données doivent être diffusés sur des dépôts publics ; ils doivent être décrits avec suffisamment de métadonnées pour que leur périmètre et la manière dont ils ont été constitués soient clairement identifiables.

1. Martin Boeckhout, Gerhard A. Zielhuis et Annelien L. Bredenoord, « The FAIR guiding principles for data stewardship : fair enough ? », *European Journal of Human Genetics*, 26–7 (juil. 2018), p. 931-936, DOI : 10.1038/s41431-018-0160-0.

2. Un exemple de ces corpus « difficiles » est le développement de la reconnaissance optique de caractères pour le Cham, langue parlée par une minorité originaire du Sud-Est de l'Asie. Celle-ci comprend environ 30000 locuteurs. Le projet travaille également à la reconnaissance du Cham ancien, qui est extrêmement méconnu (Anne-Valérie Schweyer, Jean-Christophe Burie et Tien Nam Nguyen, « Analyse, Reconnaissance et Indexation des manuscrits CHAM », dans *Documents anciens et reconnaissance automatique des écritures manuscrites*, Paris, 2022).

3. Alix Chagué et Thibault Clérice, « Sharing HTR datasets with standardized metadata : the HTR-United initiative », dans *Documents anciens et reconnaissance automatique des écritures manuscrites*, Paris, France, 2022, URL : https://hal.inria.fr/hal-03703989.

4. M. Boeckhout, G. A. Zielhuis et A. L. Bredenoord, « The FAIR guiding principles for data stewardship… », p. 931.

5. *Ibid.*, p. 932-933.

— *Accessible* : les jeux de données doivent être accessibles librement et idéalement de façon manuelle aussi bien qu'automatique.

— *Reusable* : les jeux de données doivent être réutilisables. Cela implique de définir un périmètre pour la réutilisation des données, notamment au travers de licences.

— *Interoperable* : les jeux de données doivent être diffusés dans des formats structurés, compréhensible par des humain.e.s et, idéalement, manipulables par des machines.

Qu'en est-il de la relation entre ces principes et la conception d'API ? La diffusion automatisée par le biais d'applications semble aller main dans la main avec les principes FAIR. Une API permet de diffuser automatiquement des données brutes et structurées dans des formats interopérables. Cependant, une API n'adhère pas par défaut aux principes FAIR. Elle garantit l'accessibilité des données ; pour qu'elles soient réutilisables, la conception d'API doit être considérablement étoffée. Il faut en effet définir le contexte de production des données, ainsi que la licence dans laquelle elles sont diffusées. Il est donc nécessaire d'ajouter des métadonnées aux réponses de l'API ou de trouver un moyen de documenter les conditions de réutilisation des données. La notion de « dépôt public » requis par les principes FAIR pour partager des données change également de sens. L'API sert pas à partager les données sur un autre dépôt qui agrège des jeux de données, elle sert à les diffuser à des utilisateur.ice.s. Une API pour la recherche forme donc sa propre plateforme de diffusion des données. Enfin, il faut garantir que les données soient compréhensibles. Cet aspect des principes FAIR n'est pas anodine dans la conception d'une API. Non seulement celle-ci doit renvoyer des réponses structurées et valides ; la sémantique des réponses doit également être explicitée. Cela veut dire que, soit dans l'API, soit dans la documentation, le sens de chaque élément doit être précisé. La documentation est ici importante ; mais l'utilisation de la `TEI` est également intéressante en vue de ce besoin de produire des données compréhensibles. Ce standard, en plus d'être très pratiqué dans les humanités numériques, est défini dans les *TEI Guidelines*[6] et dispose donc d'une documentation officielle. Au delà de ces principes abstraits, de véritables standards techniques peuvent servir d'inspiration dans le développement d'API pour diffuser des données de recherche.

8.2 Le REST : un modèle pour le design d'API

Ici est présenté le standard REST, défini par Roy Fielding[7] ; ce standard architectural est considéré comme l'idéal à atteindre en matière de design d'API. Il n'a cependant pas été suivi ici, en partie parce qu'il est mieux adapté à un projet de plus grande échelle, et en partie car il ne fonctionne pas de façon optimale avec le modèle de données défini,

6. TEI Consortium, *P5 : Guidelines for Electronic Text Encoding and Interchange…*
7. R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures…*

et notamment avec le choix de construire une API renvoyant des réponses conformes à la `TEI`.

Le standard REST (« État de transfert représentatif ») est un standard et un style pour la conception d'API dans une architecture client-serveur. Il est aujourd'hui un standard de fait pour la conception de tels outils, et est considéré comme une bonne pratique dans l'architecture d'API. Le standard REST ne définit pas comment une API doit être implémentée, mais définit la structure de l'interaction client-serveur. Historiquement, le développement de ce standard est fortement lié à l'établissement du protocole HTTP, puisque le concepteur du REST est l'un des ingénieur.e.s à l'origine du protocole. Le REST définit un ensemble de quatre contraintes supplémentaires à celles présentes dans le protocole HTTP [8]. Une API REST doit répondre aux contraintes suivantes :

— Être sans état [9]. Cela signifie qu'une requête du client au serveur doit contenir l'ensemble des informations nécessaires au traitement de la requête. Le serveur ne peut accéder à un contexte ou à d'autres informations pour traiter la requête. Cela permet de rendre les requêtes plus faciles à interpréter, limite le risque d'erreurs et permet à l'API de fonctionner à plus grande échelle, puisqu'elle n'a pas besoin d'utiliser de les ressources pour chercher des informations ailleurs que dans la requête.

— Présenter une interface uniforme [10]. C'est peut-être ce principe qui est définitoire du REST : l'interface, c'est-à-dire la partie de l'API qui est exposée au client, doit toujours être la même. Il y a donc un découplage entre la communication avec le client (réception des requêtes et envoi des réponse) et le traitement des requêtes, qui ne sont pas accessibles au client. Cette séparation permet de modifier la manière dont une requête est traitée sans devoir changer la sémantique et la structure de l'interaction avec le client. Ce principe implique d'autres contraintes :

  — L'identification des ressources dans des requêtes : un client requête une URI et le serveur lui renvoie une représentation de cette URI. Dans le contexte du Web, cela veut dire qu'un URL doit nécessairement être fourni au moment de la requête et que celui-ci doit pointer vers les ressources à traiter.

  — Les ressources sont communiquées sous la forme de représentations. Cela veut dire que les données envoyées au client peuvent être dans une structure ou un format différents de ceux de la base de données : l'API doit traduire les données telles qu'elles sont dans la base en leur représentation.

  — Les réponses HTTP doivent être auto-descriptives : les en-têtes et le corps de la requête doivent contenir assez de (méta-)données pour pouvoir être manipulées par le client sans avoir besoin d'informations extérieures.

8. *Ibid.*, p. 94.
9. *Ibid.*, p. 78-79.
10. *Ibid.*, p. 81-82, 86-97.

  - Utilisation de l'hypermédia comme moteur de navigation. Comme sur un site Web normal, des hyperliens doivent permettre de naviguer d'une ressource à l'autre et doivent être accessibles directement depuis l'API.

— Autoriser la mise en cache explicite [11]. Une réponse du serveur au client doit explicitement indiquer si les données contenues dans une réponse peuvent être mises en cache (c'est-à-dire, stockées dans la mémoire pour y être accédées plus tard). La possibilité de cacher les données peut potentiellement diminuer le nombre de requêtes envoyées au serveur.

— Présenter une architecture en couches [12]. Une application peut être entièrement stockée sur un serveur, ou fonctionner par l'interaction de différents serveurs. Dans ce cas, un client peut être connectés à différents serveurs, mais ne doit pas savoir s'il se trouve sur un serveur intermédiaire ou un serveur final. Les serveurs intermédiaires peuvent modifier les messages à renvoyer au client, répartir les charges sur différents serveurs ou établir des politiques de sécurité.

— Diffuser du code à la demande (optionnel) [13].

Les concepts centraux dans ce système architectural – du moins, pour une petite API comme celle de *Katabase* sont le fait d'être sans état et l'uniformité de l'interface. Ces deux concepts signifient que les requêtes comme les réponses doivent être entièrement auto-descriptives. Les autres principes sont mieux adaptés à des API à plus grande échelle : la mise en cache explicite suppose que la même requête puisse être envoyée en série par un client à un serveur ; pour une API d'un projet de recherche, il est plus probable qu'une requête soit faite une unique fois pour récupérer des données. L'utilisation d'une architecture en couche n'est pas du ressort de *KatAPI*, puisque l'intégralité de l'API est présente sur un seul serveur. Enfin, la diffusion de code exécutable n'est pas non plus adaptée aux données qui sont requêtées et diffusées via l'API.

La mise au point d'une architecture sans état demande de mettre au point une sémantique pour les requêtes qui soit auto-descriptive, c'est-à-dire qui décrive l'intégralité de la requête. L'uniformité de l'interface, d'un point de vue de développement applicatif, est une problématique plus complexe. D'abord, elle demande d'établir des modèles de données uniformes pour le corps des réponses HTTP. Cela veut dire que, pour chaque format de réponse autorisé et point d'accès aux données possible, les réponses doivent être contenues dans une structure qui est toujours la même. Ensuite, les en-têtes doivent être constitués pour décrire adéquatement la réponse. Cela passe par la définition d'un statut HTTP et d'un type MIME à chaque réponse. Ainsi, une machine peut savoir quel format de réponse est obtenu, et comment la requête s'est déroulée. Cependant, l'utilisation d'hyperliens comme moteur de l'application n'a pas été implémentée : les réponses ne

11. *Ibid.*, p. 79-81.
12. *Ibid.*, p. 82-84.
13. *Ibid.*, p. 84-85.

contiennent pas d'hyperliens qui permettent de naviguer automatiquement vers d'autres ressources ou actions. L'ensemble des ressources accessibles et la manière d'y accéder sont au contraire définies de façon centralisée dans la documentation de l'API, qui n'est pas lisible par une machine. Enfin, du point de vue de l'implémentation, la communication client-serveur est indépendante du traitement des requêtes et de l'interaction avec les bases de données.

Par conséquent, les choix techniques mis en place font de l'API une application *RESTlike* (c'est-à-dire, qui se conforme à un certain degré aux principes REST), voire *RESTful* (qui se conforme entièrement aux principes REST). *KatAPI* suit le standard REST là où il est pertinent pour les données transmises (ce qui exclut la condition de transmission de code et la clause de mise en cache). L'API suit également les principes REST là où il est de son ressort de les suivre (ce qui exclut une architecture en couches, puisque l'application est distincte de son installation sur un serveur). Suivre ces principes et s'en inspirer permet de concevoir une application au comportement uniforme, et où le client n'a pas à vérifier les réponses obtenues, puisqu'elles seront toujours les mêmes et qu'elles contiendront toutes les informations nécessaires à leur traitement (format de réponse, statut HTTP...).

8.3 OAI-PMH, CTS et DTS : quels standards pour le partage du texte en humanités numériques ?

Les principes REST, présentés ci-dessus, concernent l'architecture et le design d'une API ; cependant, ils ne s'attaquent pas à une autre partie du problème : l'interopérabilité des API. Chacune de ces applications définit sa propre sémantique pour les requêtes et autorise certains formats de réponse. Cela conduit à un problème pour les programmes consommateurs d'API : l'interaction avec deux API demande la construction de requêtes différentes, et demande donc d'apprendre à utiliser différentes API. Lorsque celles-ci sont complexes, apprendre à utiliser une API n'a rien d'anodin, et se servir de différentes API devient complexe. De plus, l'absence d'interopérabilité signifie qu'il est difficile, voire impossible, de travailler de la même manière avec les données issues d'API différentes. Si l'une renvoie du `JSON` mais l'autre du `XML`, par exemple, les résultats issus des deux API doivent être utilisés différemment. Cela complexifie la convergence des données de la recherche et limite l'interopérabilité de ces données, qui est pourtant l'un des principes FAIR [14]. Pour faire face à ce problème, des standards d'interopérabilité ont été développés. Ceux-ci définissent à minima une sémantique unique qui peut être implémentée par plusieurs API. Parfois, ils définissent également des formats de réponse. Le plus célèbre

14. M. Boeckhout, G. A. Zielhuis et A. L. Bredenoord, « The FAIR guiding principles for data stewardship… », p. 932-933.

de ces standards, dans les humanités, est le protocole IIIF, qui permet d'interagir avec une image (en lui ajoutant une couche d'annotations, par exemple). Dans les humanités centrées sur le texte, les standards d'API sont plus diversifiés. Trois sont présentés ici : l'OAI-PMH, le CTS et le DTS. Aucun d'entre eux n'a été adopté dans la conception de l'API *Katabase*, mais ils sont tous riches de leçons dans le développement d'une telle application

8.3.1 OAI-PMH : un premier standard à succès

L'OAI-PMH vient du monde des archives et des bibliothèques. Les bibliothèques ne conservant pas des objets uniques, mais des documents produits en série, elles sont pionnières dans le développement de méthodes computationnelles pour le partage des données [15]. Aussi ce standard a-t-il connu une adoption inégalée, puisqu'il est aujourd'hui le standard d'interopérabilité le plus utilisé dans les bibliothèques [16] ; il est y compris utilisé par les plateformes en ligne HAL, OpenEditions ou encore CAIRN. Ce standard vise à permettre l'interopérabilité sémantique (uniformiser la sémantique des requêtes et des réponses) et technique (uniformiser les formats de réponse) [17]. Il ne vise cependant pas à partager des documents, mais seulement leurs métadonnées (en d'autres termes, l'OAI-PMH ne permet pas de partager un document, mais seulement sa notice bibliographique) ; il ne permet pas non plus de partager des données sur un document, mais sur une collection entière. Les réponses de l'OAI-PMH sont constituées de collections de notices encodées en XML ; l'interopérabilité sémantique des réponses est permise par l'utilisation du Dublin Core, un format considéré comme un plancher pour le partage de données bibliographiques. L'interopérabilité technique, elle, repose sur un format de réponse en `XML` [18]. Comme le montrent Prime-Claverie et Mahé, cette interopérabilité n'empêche cependant pas une grande variété dans les implémentations : certaines utilisent des réponses en `XML-TEI`, d'autres non ; les structures des réponses varient elles-aussi, avec différents éléments Dublin Core associés à différentes données [19]. D'un point de vue technique, le fonctionnement d'OAI-PMH est intéressant : l'entrepôt OAI-PMH est distinct de la base d'où il récupère les données. Cela veut dire que, lorsqu'une requête est reçue, l'API la communique à l'entrepôt OAI-PMH qui transforme les informations contenues dans la base de données. D'un point de vue d'ingénierie, cela veut dire que les parties de ce système (base de données – entrepôt OAI-PMH – API) sont distincts [20]. Par conséquent, les

15. Camille Prime-Claverie et Annaïg Mahé, « Le défi de l'interopérabilité entre plates-formes pour la construction de savoirs augmentés en sciences humaines et sociales », dans *Ecrilecture augmentée dans les communautés scientifiques*, 2017, URL : `https://archivesic.ccsd.cnrs.fr/sic_01511618` (visité le 29/08/2022), p. 2-3.
16. *Ibid.*, p. 5.
17. *Ibid.*, p. 6-14.
18. *Ibid.*, p. 4.
19. *Ibid.*, p. 6-14.
20. *Ibid.*, p. 4.

données n'ont pas besoin d'être stockées dans des formats compatibles ; des transformations dans une partie n'impliquent pas de devoir transformer les autres parties (sauf pour permettre l'interaction entre les différents éléments du système). Par exemple, une modification de la structure de la base de données ne demande pas de transformer toute l'API, mais seulement la manière dont ces données sont transformées pour être compatibles avec le standard OAI-PMH.

Que retenir de cet exemple ? Tout d'abord, que ce n'est pas parce qu'un standard existe qu'il sera implémenté uniformément. Une marge de manœuvre est toujours présente, ce qui peut amener à des résultats très différents. Bien sûr, cela va à l'encontre du principe d'interopérabilité et complexifie la convergence des ressources d'origine différente. Cependant, comme le rappellent Prime-Claverie et Mahé [21], ces variations ne sont pas nécessairement problématiques. Elles permettent aux différentes institutions qui l'implémentent de fournir des données adaptées à certaines communautés de pratique – il va de soit qu'une archive et une bibliothèque sont deux institutions très différentes, et qu'elles ne peuvent représenter leurs documents de la même manière. L'implémentation technique de l'OAI-PMH est également intéressante : elle sépare les données (dans une base de données) de leur représentation. Ce qui est communiqué au client, c'est cette représentation, compatible avec le protocole OAI-PMH. Cette idée, qui rappelle les principes REST, est mise en place d'autres standards présentés ici. D'un point de vue de développement, la séparation entre les différentes parties nécessaires au fonctionnement de cet API facilitent également son maintient et l'ajout de nouvelles fonctionnalités, puisque le système ne doit pas être modifié lorsque c'est seulement un élément qui est affecté. Cependant, l'OAI-PMH n'est pas adapté aux besoins de *Katabase* : ce standard ne permet pas d'échanger du texte, mais seulement des métadonnées. De par sa taille, il est également difficile à implémenter pour un projet de recherche.

8.3.2 CTS, DTS : des méthodes de standardisation pour l'échange de texte

Le CTS a été développé dans un contexte très différent de l'OAI-PMH. Il est issu d'un projet de recherche à Harvard, le *Homer Multitext* consacré à l'édition de l'*Illiade* et de l'*Odyssée* d'Homère. Le standard est donc pensé pour l'édition de textes canoniques antiques. Il naît d'un problème technique : comment faire coïncider la structure en arbre d'une édition numérique avec la structure citationnelle traditionnelle, où les citations font références à des pages. Aux origines du projet se trouve également l'idée qu'un texte existe à l'intérieur d'un système citationnel, où les textes fonctionnent par référence les uns aux autres [22]. Cette idée est particulièrement importante dans un projet d'édition

21. *Ibid.*, p. 15-16.

22. David Neel Smith et Christopher W. Blackwell, « Four URLs, limitless apps : Separation of concerns in the Homer Multitext architecture », dans *Donum natalicium digitaliter confectum Gre- gorio*

critique, comme le *Homer Multitext*, où les différents témoins des textes d'Homère doivent pouvoir être alignés et comparés. La `TEI` a développé ses propres systèmes de référence pour les éditions critiques, basés sur le concept d'arbres décisionnels (où chaque témoin peut être représenté par une feuille, et où les différentes feuilles viennent d'une branche commune) [23]. Mais ce système ne permet pas de cibler efficacement un passage dans un texte, ni de faire référence à ce passage en dehors du document lui-même. Comment, dans ce cas, représenter des références dans une édition numérique ? Comment rendre des parties de texte accessibles à l'extérieur d'un texte, dans une architecture client-serveur ? La réponse proposée à l'initiative de Smith et Blackwell est le CTS. Ce système définit une sémantique permettant d'identifier un texte, un passage de texte ou même un ensemble de textes grâce à une URI. Celle-ci cible de plus en plus précisément le passage voulu. Par exemple, `urn:ctsl:greekLit:tlg0012.tgl001.msA` permet de cibler, à l'aide d'un identifiant CTS, le manuscrit `msA` du texte `tlg0012.tgl001` du corpus `greekLit` [24]. L'intérêt de ce système de pointeurs vers un texte est qu'il peut être adapté au Web, en précédent l'identifiant ci-dessus de l'URL d'une API supportant le standard CTS. Il est donc possible de cibler précisément et de récupérer un texte depuis un dépôt particulier. L'intérêt de ce système est également qu'il est généralisable à d'autres corpus ; le standard a donc été adapté par plusieurs API, qui ont dû définir leurs propres implémentations de ce standard : là où le standard est implémenté, il est possible d'accéder à l'aide d'une sémantique unique à n'importe quel(s) texte(s). Parmi ces implémentations peut être cité *CapiTainS*, développée pour la *Perseus Digital Library* et l'*Open Philology Project* [25]. CTS est donc intéressant d'un point de vue de l'ingénierie, puisqu'il permet la convergence des méthodes de partage et de diffusion du texte. Le standard est également intéressant d'un point de vue de la représentation du texte, puisqu'il permet de représenter un texte comme étant un graphe de citations et de références (une citation correspondant à une URI reliée à une autre par un prédicat). Ce système de représentation permet donc de s'éloigner de la structure hiérarchique des éditions numériques en `TEI`, une représentation critiquée depuis ses débuts parce qu'elle impose un ordre hiérarchique unique à un texte, pourtant irréductible à une seule interprétation [26].

Nagy septuagenario a discipulis collegis familiaribus oblatum : A Virtual Birthday Gift Presented to Gregory Nagy on Turning Seventy by His Students, Colleagues, and Friends, 2012, URL : `https://chs.harvard.edu/d-n-smith-c-w-blackwell-four-urls-limitless-apps-separation-of-concerns-in-the-homer-multitext-architecture/` (visité le 22/08/2022).

23. TEI Consortium, *P5 : Guidelines for Electronic Text Encoding and Interchange*…

24. D. N. Smith et C. W. Blackwell, « Four URLs, limitless apps : Separation of concerns in the Homer Multitext architecture »…

25. Bridget Almas et T. Clérice, « Continuous Integration and Unit Testing of Digital Editions », *Digital Humanities Quarterly*, 11–4 (févr. 2018), Publisher : Alliance of Digital Humanities, URL : `https://hal.archives-ouvertes.fr/hal-01709868` (visité le 12/07/2022).

26. Allen Renear, Elli Mylonas et David Durand, « Refining our Notion of What Text Really Is : The Problem of Overlapping Hierarchies », dans *Research in Humanities Computing*, dir. Nancy Ide et Susan Hockey, Oxford, 1996, URL : `https://cds.library.brown.edu/resources/stg/monographs/ohco.html` (visité le 22/08/2022).

Le standard CTS n'est cependant pas parfait, et plusieurs critiques peuvent lui être faites. D'un point de vue de son développement, c'est un standard qui a été développé dans une certaine communauté de pratique, et qui n'est donc pas adaptable à tous les textes ; ensuite, c'est un format qui a été développé pour répondre aux besoins de chercheur.euse.s, et qui ne suit donc pas les meilleures pratiques en termes de conception d'API. Par exemple, il ne définit pas de formats de données à utiliser pour la réponse à renvoyer à l'utilisateur.ice [27]. La `TEI` n'est donc pas requise, peut-être de la critique faite par Smith et Blackwell des représentations hiérarchiques du texte. Si cela peut avoir un intérêt de ne pas vouloir « enfermer » le texte dans une hiérarchie qui lui est étrangère, ne pas utiliser la `TEI` pour une édition nativement numérique aujourd'hui est un choix pour le moins surprenant : ce standard dispose d'une très grande communauté d'utilisateur.ice.s, et est au cœur des humanités numériques centrées sur le texte. Le fait de ne pas imposer de format unique amène à beaucoup de variété dans les implémentations, ce qui s'oppose à un objectif de convergence des initiatives et ne garantit pas l'interopérabilité technique entre les API. De fait, les implémentations de CTS varient beaucoup, comme le montre l'exemple de *CapiTainS*, qui lui rend obligatoire l'utilisation du format `XML` dans les réponses. C'est pourquoi un nouveau standard a été développé : le DTS. Alors que le CTS existe depuis une décennie, la première version publique du CTS a été publiée en 2018. Elle est basée sur le même principe que le standard précédent : une sémantique standardisée permet à une URI de représenter un ou plusieurs (extraits) de texte. Plusieurs ajouts ont été faits à son prédécesseur. Le plus notable est l'obligation d'utiliser le `XML-TEI` [28], ce qui garantit l'interopérabilité des ressources distribuées. Le standard cherche également à intégrer les meilleures pratiques en matière de développement Web, en utilisant des formats sémantiques (comme le `JSON-LD`) et en encourageant le développement d'API RESTful. Ce standard est également considérablement plus complexe que le CTS puisque l'API peut offrir plusieurs points d'accès aux texte ; ces points d'accès correspondent à différents niveaux de la collection (allant de la collection dans son ensemble au document). Bien que récent, le format a été implémenté par plusieurs projets, dont *TEI Publisher*, une application en ligne visant à publier des données de recherche en `TEI`.

Ces deux standards sont intéressants, mais ne sont pas adaptés à *KatAPI*. D'abord, leur implémentation complexifie considérablement la conception d'une API, et son utilisation est peut-être plus adaptée à des initiatives faisant converger plusieurs projets. Mais le principal problème posé par les trois standards présentés ici est qu'ils sont tous développés pour partager des textes réels, qui ont une existence « en dehors de l'API ». Or, *KatAPI* n'est pas seulement censée partager des textes issus du projet (c'est-à-dire,

27. B. Almas, T. Clérice, Hugh Cayless, V. Jolivet, Pietro Maria Liuzzo, Matteo Romanello, Jonathan Robie et Ian W. Scott, *Distributed Text Services (DTS) : a Community-built API to Publish and Consume Text Collections as Linked Data*, mars 2021, URL : `https://hal.archives-ouvertes.fr/hal-03183886` (visité le 01/01/2022), p. 2.

28. *Ibid.*, p. 3.

des catalogues ou des extraits de ceux-ci). Elle doit tout aussi bien partager des données de recherche issues du projet et construire des extraits du corpus en fonction des requêtes des utilisateur.ice.s. Cette API repose donc, comme nous le verrons, sur la création de documents *ex-nihilo* à partir de l'agrégation et de la transformation d'autres documents. Les critères bibliographiques servant à situer texte ou extrait dans une collection ne sont donc pas entièrement pertinents ici. L'API doit fonctionner comme un moteur de recherche plutôt que comme un catalogue dans lequel il est possible de sélectionner une ou plusieurs ressources. Cependant, les standards CTS et DTS présentent des bonnes pratiques, et de nombreux enseignements sont à en tirer. D'abord, il est nécessaire de proposer une réponse en `XML-TEI`, puisque ce standard de fait peut permettre de faire converger des données issues de différents projets. De CTS ressort l'intérêt de proposer plusieurs points d'accès au corpus, qui correspondent à différentes échelles dans celui-ci. Il n'existe pas une seule voie d'accès pour un texte, et c'est pourquoi ce principe sera suivi pour *KatAPI*. De CTS et DTS, il faut également retenir l'utilisation de l'URL comme d'un pointeur, à la sémantique clairement définie, vers les ressources pertinentes (un principe déjà énoncé dans le protocole HTTP). Enfin, des trois standards présentés ici, il faut retenir le principe de la *separation of concerns* (« séparation des préoccupations »). Ce principe central du développement applicatif est très bien exemplifié dans le fonctionnement d'OAI-PMH : la base de donnée est distincte de l'entrepôt OAI-PMH qui est lui-même distinct de la partie de l'API chargée de recevoir des requêtes et de transmettre des réponses au client. Ce principe permet de minimiser l'impact d'un processus sur un autre et de réduire les relations entre les parties d'un programme à un strict minimum. C'est un concept qui a été suivi dans l'architecture de *KatAPI*, où la réception et l'analyse des requêtes, la récupération des données, la construction des réponses et l'envoi de celle-ci à l'utilisateur.ice sont distinctes.

8.4 Pour qui sont ces API ? Qu'est-ce qui est demandé d'un tel outil ?

Cette dernière question n'est pas anodine. Il semblerait en effet que relativement peu d'API soient développées dans des contextes d'humanités numériques ; celles qui sont développées sont plutôt le fait de grandes institutions – comme le projet *Europeana*, ou la Bibliothèque nationale de France – et de plateformes partagées qui adoptent le standard OAI-PMH. Il existe également de nombreux services nécessitant l'utilisation d'API, comme les serveurs IIIF permettant le partage d'image, et les serveurs DTS qui commencent à se développer. La conception d'API semple être le fait d'infrastructures qui disposent de grands volumes de données et de ressources permettant de planifier et d'implémenter de tels outils sur le long terme.

Une étude réalisée dans le cadre du projet *Europeana* aide à situer le statut des API dans les humanités numériques. Cette étude offre un point de vue intéressant, puisqu'elle s'appuie sur des entretierequêtens avec des chercheur.euse.s issue.e.s des humanités numériques et des humanités s'appuyant sur des méthodes quantitatives. Cette étude arrive à un constat pessimiste : les chercheur.euse.s ne sont pas des utilisateur.ice.s d'API[29]. Même lorsque leurs recherches s'appuient sur le traitement de données, voire sur l'usage de méthodes computationnelles (traitement statistique à l'aide de `python` ou `R`), ces personnes ont tendance à récupérer leurs données manuellement ; ce qui leur importe, c'est les données, et non la manière d'y accéder[30]. Les API sont, à l'inverse, beaucoup plus utilisées par des ingénieur.e.s et des personnes ayant eu une formation technique. Au delà de ce constat, l'étude propose des réflexions intéressantes sur le statut des API dans les humanités en général et numériques en particulier. Trois critères sont identifiés pour la décision d'utiliser une source de données en recherche[31] :

— Les données. Le critère de choix principal est la qualité et la complétude des données et des métadonnées.

— L'expertise technique nécessaire. Ce critère est relativement logique : un.e chercheur.euse n'utilisera une API que si il ou elle a le moyen de le faire. Un problème ici est la méconnaissance des API par les chercheur.euse.s, qui peuvent ne même pas connaître l'existence de tels outils. Cependant, l'étude remarque que, une fois qu'ils ont connaissance de telles sources, les chercheur.euse.s n'hésitent pas à se former à l'utilisation d'API.

— Les environnements sociaux et techniques des chercheur.euse.s. Le critère précédent décrit des capacités réelles, alors que celui-ci traite de la perception. Plus un.e chercheur.euse perçoit que des outils techniques sont difficiles d'accès, moins il ou elle est susceptible de les utiliser.

Pourquoi, alors, développer une API ? Au vu du volume du corpus, il peut être intéressant d'y avoir accès de façon automatisée. Avant le développement de cette API, la seule manière d'accéder à des données brutes était de les télécharger sur les dépôts GitHub du projet. Dans ce cas, il est uniquement possible de récupérer un fichier complet, c'est-à-dire un catalogue entier ou un jeu de données portant sur l'ensemble du corpus. La seule manière d'accéder à des données sélectionnées et filtrer en fonction de ses besoins est de passer par l'application Web « pour huamin.e.s », mais celle-ci ne permet pas de télécharger les données brutes. Au sein du panorama d'outils développés pour le projet, les méthodes de diffusion de données manquaient. C'est pourquoi une API a été développée.

29. Jennifer Edmond et Vicky Garnett, « APIs and Researchers : The Emperor's New Clothes ? », *International Journal of Digital Curation*, 10–1 (21 mai 2015), p. 287-297, DOI : `10.2218/ijdc.v10i1.369`, p. 288.

30. *Ibid.*, p. 290.

31. *Ibid.*, p. 292-294.

Plus largement, les API peuvent être particulièrement utiles pour un projet de recherche et pour l'application des principes FAIR en humanités numériques. Une API diminue, sur le long terme, le travail nécessaire pour diffuser des données. Dans le cas d'une API « réactive », qui construit des documents sur mesure, elle permet également d'explorer des sous-catégories, ou des sections du corpus. Une telle API peut permettre la création de nouveaux jeux de données à la demande, ce qui est d'un intérêt certain.

L'étude d'Edmond et Garnett ne doit pas être prise avec pessimisme, indiquant que l'outil que l'on développe ne sera jamais utilisé. Au contraire, les critères sociaux-techniques qu'elles mettent en avant derrière l'utilisation d'une API permettent de mieux adapter l'outil développé aux besoins. Il est nécessaire, lorsque cela est possible, de développer un outil qui diminue la distance perçue par les chercheur.euse.s. Pour cela il faut documenter l'application, de façon claire et, lorsque cela est possible, concise. Ensuite, il faut chercher à montrer que l'utilisation d'une API n'est pas très compliquée : il peut être intéressant de concevoir des tutoriels, ou comme cela a été fait pour *KatAPI*, des méthodes pour tester l'API en temps réel. Depuis la page de documentation, il est possible de composer un dictionnaire contenant les paramètres pour lancer une requête. Une fois la réponse reçue, le résultat s'affiche sur la page. De cette étude, il faut également retenir que une API ne sera utilisée que si elle offre l'accès à des données de qualité. Cela va dans le sens des principes FAIR : pour partager des données, celles-ci doivent être documentées et éditorialisées pour qu'un.e chercheur.euse sache quelles données seront reçues et envisage des manières de les utiliser.

Une autre étude[32], menée sur des API généralistes, permet également de mieux comprendre le terrain pour de tels outils. Cette étude est basée sur l'analyse statistique des réponses obtenues obtenues pour 10955 *Buisness APIs*, terme qui recoupe des applications développées par des entreprises (comme l'API de *Twitter*), à but commercial (comme l'API de *PayPal*), mais aussi des API créées à but non-lucratifs (comme celle de *Wikipedia*). L'étude date d'il y a quelques années, et il est donc possible que les tendances de design d'API aient évoluées depuis. Les statistiques coincident cependant avec mon expérience de ces outils. Les résultats obtenus permettent de mieux comprendre quelles sont les tendances actuelles en terme de design d'API. Les deux formats de réponse dominants sont le `JSON` et le `XML` : 81,08% des applications proposent l'un ou l'autre format. Seules 21,37% utilisent les deux formats, ce qui est le cas de *KatAPI*. Un autre fait intéressant est que une immense majorité d'API sont *RESTlike*[33]. Contrairement aux API *RESTful*, les API *RESTlike* sont inspirées par les principes REST sans nécessairement les appliquer

32. Andrea Janes, Tadas Remencius, Alberto Sillitti et Giancarlo Succi, « Towards Understanding of Structural Attributes of Web APIs Using Metrics Based on API Call Responses », dans *Open Source Software : Mobile Open Source Technologies*, dir. Luis Corral, Alberto Sillitti, Giancarlo Succi, Jelena Vlasenko et Anthony I. Wasserman, Series Title : IFIP Advances in Information and Communication Technology, Berlin, Heidelberg, 2014, t. 427, p. 83-92, DOI : 10.1007/978-3-642-55128-4_11.

33. *Ibid.*, p. 3.

totalement. Ce terme s'est développé du fait de l'implémentation très inégale du standard. Le problème est que le terme *RESTlike* est nécessairement imprécis ; il est donc difficile de tirer une conclusion de cette statistique : dans une large mesure, les principes REST ont servi au développement de *KatAPI*. Quoi qu'il en soit, *KatAPI* semble s'inscrire, par les formats utilisés et les principes architecturaux suivis, dans le paysage actuel des API.

Les standards et études présentés ici permettent de mieux cerner le statut des API centrées sur le texte en humanités numériques ; elles permettent également d'identifier des tendances dans le desgin d'API ainsi que les besoins des utilisateur.ice.s. En ayant cette compréhension du paysage technique pour ces applications, il est possible de définir clairement le périmètre de l'API et la manière dont l'interaction client-serveur doit s'organiser, dans la sémantique des requêtes autant que dans les réponses obtenues.

Chapitre 9

Définir un périmètre : que partager, et comment partager ?

Après avoir présenté le paysage des API en général et le statut de ces outils dans les humanités numériques en particulier, ce chapitre décrit *KatAPI* du point de vue de l'utilisateur.ice : quelles données peuvent être obtenues via l'API, quels paramètres de recherche sont autorisés et quels sont les formats de données retournés par l'application. C'est donc les principes de la communication client-serveur de *KatAPI* qui sont ici présentés.

9.1 Grands principes pour l'architecture de l'API

Pour plus de clarté, ici sont présentés de façon concise l'ensemble des principes auxquels l'application doit se conformer ; ceux-ci forment une sorte de cahier des charges pour son implémentation technique.

— L'application supporte uniquement la méthode HTTP `GET` : il est uniquement possible de demander des données à l'application, et non d'ajouter des données à une base de données, par exemple.

— L'application supporte deux formats de réponse : `JSON` et `XML-TEI`. Le format par défaut est le `JSON`, pour sa légèreté et sa facilité de manipulation. Dans les deux cas, la structure des réponses est la même : un en-tête qui décrive le contexte de la réponse (à ne pas confondre avec les en-têtes HTTP) et un corps, qui contient les données récupérées.

— Trois niveaux d'accès aux données sont possibles. Il est possible de faire une requête pour un catalogue entier ; des statistiques sur un ou plusieurs catalogues ; une ou plusieurs entrées de catalogues.

— L'application suit le principe de séparation des préoccupations. L'interaction avec le client, le traitement des requêtes et la base de données sont des parties distinctes qui ne communiquent que lorsque cela est nécessaire.

— L'API suit les principes REST lorsque cela est possible. Elle est sans état et présente une interface uniforme. Cela veut dire que, pour les trois niveaux d'accès aux données, trois modèles de données seulement existent et sont adaptés en `JSON` et `XML-TEI`.

— L'application a un traitement strict des requêtes. Une requête n'est pas traitée si elle contient des paramètres non-autorisés (qui ne font pas partie de la sémantique définie pour l'interaction client-serveur), ou des valeurs associées à ces paramètres qui ne sont pas autorisées.

— L'application envoie des réponses précises et auto-descriptives, même en cas d'erreur. Plusieurs statuts HTTP sont définis pour différents cas de figures ; si erreur il y a, un message d'erreur complet et décrivant le contexte et les raisons de l'erreur est renvoyé par l'API. Cette erreur constitue un document `JSON` ou `XML-TEI` valide.

9.2 Quelles données partager ?

À différentes étapes, une grande variété de données ont été produites ; de plus, différentes manières d'accéder aux données ont été constituées par le projet. Par exemple, un système de réconciliation des différentes entrées de catalogue permet d'identifier lorsqu'un manuscrit est vendu plusieurs fois, et donc présent dans différents catalogues. Cela permet de regrouper ensemble différentes occurrences d'un même manuscrit. Ce type de données pourrait être partagé, de même que les données extraites de `SPARQL` et présentées dans la partie précédente. Cependant, la mise à disposition de différents types de données est difficile tout en se conformant au principe d'uniformité de l'interface du REST. En effet, chaque source de données a sa propre structure. Il donc est nécessaire, pour chaque source de données, de définir une représentation en `JSON` et en `TEI`. Pour se conformer à l'objectif originel du projet *MSS / Katabase* (étudier des catalogues en vente), il a été choisi de ne diffuser que des données issues des catalogues à travers l'API. Cela élimine donc les données issues de `SPARQL`. Pour ne se conserver qu'une seule représentation par type de données, il a été choisi de ne pas diffuser non plus de données sur les manuscrits « réconciliés », puisque cela amènerait à avoir deux modèles de données pour les manuscrits, selon qu'ils aient été réconciliés ou non. De plus, ce processus de réconciliation prend quelques secondes à s'exécuter, ce augmente le temps de réponse de l'API et peut devenir problématique si de nombreuses requêtes sont reçues en même temps pas l'application.

Au final, trois niveaux d'accès aux corpus ont été définis. Ceux-ci correspondent à différents degrés de granularité dans le corpus. Chacun de ces formats, enfin, a un modèle de données qui lui est propre. Celui-ci est transposé à la fois en `XML-TEI` et en `JSON`.

— Premier niveau : catalogue complet (`cat_full`) : l'intégralité des données contenues dans un catalogue peut être requêtée. Dans ce cas, il n'est possible que de requêter

un catalogue à la fois. Celui-ci doit être clairement identifié à l'aide de son identifiant unique (l'`@xml:id`). Il est renvoyé dans son intégralité au format `XML` uniquement : la traduction d'un fichier `XML-TEI` complet en `JSON` est complexe ; de plus, si un.e utilisateur.ice demande un catalogue de façon aussi ciblée, alors il ou elle souhaitera probablement un format plus complexe que le `JSON` et pourra tirer parti du balisage sémantique permise par la `TEI`.

— Deuxième niveau : statistiques sur une collection de catalogues (`cat_stat`) : des données statistiques sont retournées pour un ou plusieurs catalogues (prix moyen et médian des items, variance des prix au sein du catalogue, nombre d'items en vente...). En utilisant d'autres paramètres de recherche, il est possible de cibler précisément un sous-ensemble de catalogues, comme ceux issus de la *Revue des autographes*, ou ceux publiés dans une certaine tranche de dates. Ce degré de granularité permet donc de produire des sous-ensemble pour des études statistiques et thématiques ciblées.

— Troisième niveau : au niveau de l'entrée (`item`). Ce degré permet d'accéder à une ou plusieurs entrées de catalogues, et donc de récupérer l'ensemble des informations contenues dans les catalogues pour ces entrées. Il est possible de cibler des entrées par nom auteur.ice, date d'écriture, date de vente ou par identifiant `@xml:id`, ce qui permet de ne retourner qu'un identifiant. Ce degré de granularité permet donc de constituer des jeux de données restreints, avec par exemple tous les manuscrits écrits par un.e auteur.ice à une certaine période.

Chacun de ces degrés d'accès aux données correspond à une source de données différente. Le premier niveau (`cat_full`) correspond à un document `TEI` représentant un catalogue dans son intégralité. Ce format permet donc l'accès aux sources directes du projet. Le deuxième niveau (`cat_stat`) donne l'accès à un `JSON` produit à partir d'une analyse statistique de tous les catalogues. Les entrées pertinentes de ce `JSON` sont traduites en `XML-TEI` si c'est le format de requête qui est demandée par l'utilisateur.ice. Enfin, le niveau `itm` correspond à deux sources de données : la première, c'est les catalogues eux-mêmes (au niveau de l'entrée, et non du catalogue complet) ; la deuxième, c'est une représentation en `JSON` des entrées de catalogues. Pour accélérer le fonctionnement de l'API lorsque la requête est faite au niveau de l'item, les données en `JSON` sont d'abord lues avant, si besoin, d'utiliser la `TEI`. Les trois degrés de granularité des données sont donc centraux à l'architecture de l'API : ils définissent trois bases de données à utiliser ; ils déterminent aussi trois représentation de données différentes (c'est-à-dire, trois structures pour les réponses de l'API). Grâce au principe de séparation des préoccupations, il est possible de rendre plus tard d'autres données accessibles et de permettre donc la diffusion de données depuis de nouvelles sources et la construction de nouvelles représentations. Les degrés de granularité déjà définis ne seront pas transformés par l'ajout d'autres degrés

de granularité pour l'accès au corpus. Puisque les données et leur représentation sont distinctes, il est également possible de modifier la structure des sources de données (la structure des catalogues, par exemple) sans que le format de sortie ne soit affecté. Il suffit pour ce faire de faire en sorte que la représentation des données soit la même malgré le changement de la source de données.

9.3 Codifier l'accès aux données : une sémantique pour les requêtes faites à l'API

Les trois niveaux présentés au dessus sont autant de points d'accès aux données. Ils définissent les degrés de précision possibles pour accéder au données, mais ne définissent pas entièrement comment récupérer les informations voulues. À cette fin, une sémantique complète a été définie pour accéder aux données. Celle-ci s'organise autour des trois points d'accès présentés ci-dessus, puisque certains paramètres peuvent être interdits, ou que ceux-ci acceptent différentes valeurs en fonction du point d'accès aux données. Les paramètres de recherche autorisés sont les suivantes :

— `level` : il s'agit du niveau auquel faire une requête. C'est ce paramètre qui définit le point d'accès aux données, et le degré de précision autorisé pour les recherches.

— `format` : le format de la réponse renvoyée par le serveur au client.

— `id` : rechercher un catalogue ou une entrée de catalogue par son identifiant.

— `name` : rechercher un catalogue par type (*Revue des autographes*, vente aux enchères...) ou une entrée de catalogue par nom d'auteur.ice.

— `sell_date` : la date de vente à laquelle correspond un catalogue, ou la date de vente d'un manuscrit.

— `orig_date` : la date d'écriture d'un manuscrit.

Ces cinq paramètres offrent un vocabulaire commun pour accéder à toutes les données issues des catalogues. Pour une API, avoir une sémantique claire et facile d'accès est important, et c'est pourquoi les paramètres de recherche sont limités à ces cinq mots clés. Le problème est que ces cinq mots clés permettent d'accéder à différents types de données, en fonction du point d'accès choisi avec le paramètre `level`. Deux grands types peuvent être identifiés : les données portant sur un catalogue et celles portant sur un manuscrit. Ces deux catégories portent cependant sur des objets très différents : bien que la sémantique soit la même pour toute l'API, il n'est pas possible de rechercher un catalogue et un manuscrit dans les bases de données disponibles. Cette sémantique doit donc être définie et adaptée aux différents points d'entrée aux données.

Comme cela a été dit, l'API réalise un contrôle strict des requêtes : une requête n'est faite que si elle correspond à la sémantique des requêtes ; si elle contient des pa-

ramètres interdits, alors la requête n'est pas lancée et un message d'erreur est renvoyé à l'utilisateur.ice. Cela permet d'économiser les ressources du serveur : chaque requête a un certain prix. Celui ci peut s'exprimer en terme de temps (le traitement d'une requête n'étant pas instantané) et en termes computationnels (la charge processeur nécessaire à traiter une requête, c'est-à-dire rechercher des informations dans plusieurs bases de données, construire des réponses et les renvoyer à l'utilisateur.ice). Par ailleurs, certaines requêtes pourraient correspondre à la sémantique définie, mais n'avoir aucun sens pour le moteur de recherche : si un identifiant (`id`) et un nom (`name`) sont fournis, laquelle de ces deux informations utiliser pour identifier les items pertinents ? Un nom et un identifiant peuvent en effet pointer vers deux ressources très différentes. Le parti pris a donc été de diminuer les coûts de traitement et le risque de requêtes contradictoires en contrôlant strictement celles-ci. D'un point de vue de sécurité informatique, le contrôle des requêtes peut également protéger contre les attaques par déni de service, qui consistent à faire énormément de requêtes au même moment pour faire surcharger un serveur et l'empêcher de toutes les traiter.

La sémantique des requêtes ne dépend donc pas seulement des mots clés, mais aussi des valeurs qui y sont associées. Les principales variations dépendent du point d'accès aux données choisi. Cependant, dans tous les cas, une valeur doit être fournie pour le paramètre `id` ou `name`, puisque c'est à partir de ceux-ci qu'est faite une recherche. Au niveau du catalogue complet (`cat_full`), le contrôle des requêtes est très précis, puisque les deux paramètres autorisés sont `format` et `id`.

— Le seul format de réponse autorisé (paramètre `format`) est la `TEI`, pour deux raisons. D'un point de vue scientifique, un catalogue encodé en `TEI` contient des informations qui ne peuvent être facilement représentées sous la forme d'un `JSON`. D'un point de vue scientifique, les catalogues en `TEI` sont le fruit d'un travail éditorial particulier ; il est considéré qu'un.e utilisateur.ice qui fasse une requête aussi précise souhaiterait disposer d'une représentation aussi précise que la `TEI`. D'un point de vue computationnel, la représentation en `JSON` d'un catalogue complet encodé en `TEI` serait très difficile, et aboutirait certainement à une perte d'informations.

— À part `format`, le seul autre paramètre autorisé est `id`. Une requête au niveau du catalogue complet ne peut renvoyer qu'un seul de ces documents à la fois, et essayer de cibler un catalogue par son nom, ou sa date de publication risquerait de créer des doublons. Là encore, il est supposé qu'un.e utilisateur.ice voulant obtenir un catalogue complet sait précisément celui que il ou elle cherche, et pourrait donc fournir directement son identifiant. Cela dit, il est tout à fait possible de récupérer plusieurs catalogues en lançant une série de requêtes avec différents identifiants. Suivant le principe de contrôle strict des requêtes, seul un identifiant valide sera recherché. Un identifiant valide correspond à l'expression régulière `CAT_\d+`, soit `CAT_` suivi d'un nombre entier (par exemple, `CAT_000001, CAT_000499`).

Lorsque des données statistiques sur des catalogues sont recherchées (au niveau `cat_stat`), l'API offre plus de liberté dans les paramètres fournis.

— Le `format` de réponse accepte deux valeurs possibles : `tei` ou `json`.

— Un `id` ne peut être fourni en même temps qu'un `name`, pour éviter des requêtes impossibles à traiter.

— Un identifiant doit toujours être conforme à l'expression régulière `CAT_\d+`, afin de toujours identifier un catalogue.

— Le paramètre `name` n'accepte que certaines valeurs. Celles-ci correspondent aux différents types de catalogues établis par le projet : `RDA` pour ceux issus de la revue des autographes, `AUC` pour les ventes aux enchères...

— Le paramètre `orig_date` n'est pas autorisé : au niveau d'un catalogue complet, il n'y a pas de date de création, mais seulement la date de la vente à laquelle correspond le catalogue.

— Le paramètre `sell_date` n'accepte que des valeurs au format `AAAA` ou `AAAA-AAAA`, ce qui est validé par l'expression régulière `\d{4}(\d{4})+`.

C'est au niveau de l'item que la plus grande liberté est admise pour les paramètres de recherche :

— Le `format` de réponse peut être du `XML-TEI` ou du `JSON`.

— Une entrée de catalogue peut être ciblée par son identifiant avec le paramètre `id`. Cependant, l'identifiant doit correspondre au format de notation défini par *MSS / Katabase* en se conformant à l'expression régulière `CAT_\d+_e\d+_d\d+`. Des valeurs autorisées sont par exemple : `CAT_000069_e2_d1, CAT_000420_e49_d1`.

— Les paramètres `sell_date` et `orig_date` sont tous les deux autorisés et peuvent être utilisés en même temps, puisqu'un manuscrit a à la fois une date de création et une date de vente. Le format de date autorisé est le même que pour le niveau `cat_stat`. Cela permet par exemple de cibler un manuscrit écrit à une période et vendu une certaine année.

À partir de seulement cinq paramètres, c'est un moteur de recherche à filtre qui est défini et qui permet de définir la granularité des données à traiter, leur période et le format obtenu. Il est donc possible, en tant qu'utilisateur.ice de l'API, de choisir entre la facilité de traitement (avec une réponse en `JSON`) et un format à la sémantique clairement documentée (la `TEI`). Que les données soient retournées en `TEI` ou en `JSON`, la précision des résultats obtenus est la même, puisque les deux formats de réponse sont des représentations des mêmes données, en accord avec les principes REST. Une sémantique définie aussi précisément a ses avantages : elle permet de ne faire que des recherches qui ont un sens. Définir une sémantique claire et des incompatibilités entre différents paramètres de recherche garantissent que l'API pourra traiter toutes les requêtes sans risque de contradiction.

Le contrôle des données envoyées par le client permettent également de ne faire que des recherches qui permettront d'obtenir un résultat, ce qui est également plus intéressant pour les utilisateur.ice.s. Cependant, elles augmentent la difficulté d'utilisation de l'API. Cela risque d'augmenter la niveau de technicité perçu par un.e chercheur.euse, ce qui risque de les décourager d'utiliser l'outil [1]. Pour diminuer cette distance perçue, quatre solutions ont été définies :

— Des valeurs par défaut ont été établies pour certains paramètres. Par exemple, lorsqu'un `format` n'est pas spécifié par l'utilisateur.ice, la réponse est en `JSON` par défaut (sauf si c'est un catalogue entier qui est recherché). Cela permet d'omettre des paramètres, et donc de faciliter l'utilisation de *KatAPI*. La définition de valeurs par défaut semble aller à l'encontre de l'architecture « sans-état » voulue par le REST, mais de nombreuses autres API suivent ce principe (celle de *Wikidata*, par exemple).

— La recherche d'un nom est insensible à la casse, aux accents et à la ponctuation. Les résultats obtenus en recherchant le `name SÉVIGNÉ` seront les mêmes que pour `sevigne` et `Sévigné`.

— Un système d'essai en temps réel, depuis la version « pour humain.e.s », du site a été mis au point. Il est possible d'écrire un `JSON` contenant les paramètres voulus dans une barre de recherche pour faire une requête. Celle-ci est traitée par le serveur qui retourne une réponse, et celle-ci s'affiche sur la même page Web. Cela permet d'essayer une API – qui par définition n'a pas une interface graphique, ce qui est rebutant pour un.e utilisateur.ice non formé.e aux méthodes computationnelles – depuis une interface graphique.

— En cas d'erreur, les réponses identifient précisément l'erreur et permettent de corriger sa requête sans devoir consulter la documentation, comme cela est expliqué dans la section suivante.

9.4 Quelles représentations pour les données ? Principes suivis pour le partage d'informations, formats et structure des réponses de l'API

Une API n'a de sens que si elle est clairement documentée : c'est la seule manière de garantir que son fonctionnement soit clairement compréhensible par l'utilisateur.ice. La sémantique présentée ci-dessus est donc également documentée sur le site *Katabase*. Mais la sémantique d'une API ne se limite pas dans les paramètres de recherche ; elle définit également les formats de réponse de l'application. En effet, si ceux-ci ne sont

1. J. Edmond et V. Garnett, « APIs and Researchers… », p. 292-294.

pas clairement compréhensibles par un.e humain.e, ils risquent d'être inutilisables. La définition de formats de réponse est donc une sorte de travail éditorial, qui doit répondre à trois besoins :

— Les réponses doivent être structurées et manipulables par des machines. C'est le cas de la `TEI` comme du `JSON`, qui disposent tous les deux d'une structure hiérarchique en arbre permettant de clairement distinguer les différentes parties de la réponse.

— Les réponses doivent être des représentations uniformes de données, en accord avec les principes REST. Cela veut dire que, pour les différents points d'accès aux données, un seul modèle doit exister et qu'il doit être le même, peu importe le nombre de résultats ait été retourné par l'application. En accord avec ces principes, les réponses doivent également être auto-descriptives.

— La réponse doit correspondre aux principes FAIR autant qu'aux attentes des chercheur.euse.s et être compréhensible et utilisable par des personnes humaines.

C'est le troisième point qui est le plus problématique : pour qu'un fichier soit manipulable par une machine, il suffit qu'il présente une structure précise. Pour qu'une réponse ait du sens, qu'elle doit compréhensible par une personne humaine, la tâche est cependant plus compliquée. En effet, une réponse d'une API doit, en général, être aussi légère et concise que possible, puisqu'elle doit être envoyée à un client qui se trouve à distance. Une réponse « lourde » est également plus difficile à traiter par une machine. Une personne humaine risque également d'être découragée par une réponse trop complexe, qui présente par exemple des structures trop imbriquées et donc des éléments difficiles d'accès. Cependant, il n'est pas possible de demander à l'utilisateur.ice de se référer systématiquement à une documentation externe pour comprendre ce que l'application lui a renvoyé. Cette situation est encore complexifiée lorsque l'on essaye d'adhérer aux principes FAIR et aux besoins de la recherche. En effet, ces principes autant que les chercheur.euse.s mettent l'accent sur l'importance des métadonnées dans le choix d'une source de données plutôt qu'une autre [2]. Il est bien sûr possible de faire figurer toutes les métadonnées nécessaires dans la documentation de l'API qui est accessible en ligne. Cependant, cela demande de devoir régulièrement consulter une source externe ; si les résultats d'une requête sont transmis de manière brute à d'autres personnes, il est également nécessaire que celles-ci disposent des métadonnées nécessaires à leur utilisation. Il est donc intéressant d'inclure celles-ci dans la réponse renvoyée par l'API, quitte à ce qu'elles soient ignorées par l'utilisateur.ice. Deux types de métadonnées peuvent être définies : celles qui concernent les données originelles et celles qui définissent le contexte de récupération des données et de production des ressources (c'est-à-dire, la requête faite à l'API). Dans la première catégorie se classent la définition de la licence sous laquelle les données sont diffusées et la définition des termes

2. *Ibid.* ; M. Boeckhout, G. A. Zielhuis et A. L. Bredenoord, « The FAIR guiding principles for data stewardship… », p. 290-291 et p. 932-933 respectivement.

spécifiques au projet qui pourraient être présents dans les réponses ; dans la deuxième catégorie se placent les métadonnées qui permettent de définir le contexte de production de la ressource, c'est-à-dire la date de la requête, son contenu et le statut HTTP de la réponse. I va cependant de soit qu'il est impossible de contenir toutes les métadonnées définissant le projet *Katabase*, le contexte de production des données et les choix faits pendant leur traitement. Ces informations sont notamment contenues dans le `teiHeader` (l'en-tête) des catalogues encodés. Il est donc pertinent de consulter la documentation de l'API et les catalogues encodés pour mieux comprendre les données présentes dans la réponse. Cependant, une réponse doit contient toutes les données nécessaires à sa compréhension minimale, même par une personne autre que celle qui a lancé la requête. Pour faire coïncider ce besoin la nécessité d'une réponse compréhensible par une machine et qui corresponde au principe REST d'uniformité de l'interface, c'est donc une véritable éditorialisation des données à renvoyer au client qui est nécessaire.

9.4.1 Les réponses, en général

Les formats de réponse varient en fonction de plusieurs facteurs : le point d'accès aux données, le format de réponse demandé par le client et le fait qu'il y ait eu une erreur. Tous ces facteurs amènent à des structures de données qui sont différentes. Ce sont des représentations. Cependant, à un niveau abstrait et indépendant de sa représentation, une réponse peut toujours être représentée de la même manière (9.1) [3] :

Toutes les réponses revoyées par l'API contiennent un en-tête HTTP et la réponse elle-même. L'en-tête HTTP sert à caractériser la réponse à l'aide de métadonnées. Ils contiennent de nombreuses informations qui permettent d'interpréter la réponse, dont la plupart sont produits automatiquement : par exemple, des données sont produites sur le serveur ayant renvoyé la réponse, la possibilité de mettre les données reçues par le client en cache ou encore la date de la requête. Parmi toutes ces informations, deux sont définies par l'API à chaque requête : le statut HTTP (qui indique comment s'est déroulée l'interaction avec le serveur) et le type MIME (qui identifie le format de la réponse). Ainsi, les en-têtes permettent à un client d'interpréter le message reçu en réponse.

La réponse est elle-même séparée entre un en-tête et un corps. Le premier définit le contexte dans lequel la réponse avec le serveur a été faite. Cet en-tête a plus de précision que celui prévu par le protocole HTTP ; contrairement à celui-ci, qui n'est plus accessible une fois l'interaction avec le serveur terminée, l'en-tête de la réponse est toujours accessible, puisqu'il fait partie du fichier renvoyé. Le corps de la réponse en lui-même contient les données correspondant à la requête du client, ou, si une erreur a eu lieu, un

3. Des exemples de réponses en différents formats et avec différents paramètres sont visibles en annexes : au niveau `cat_stat` en `JSON` (20) et `TEI` (20), au niveau de l'`item` en `JSON` (18) et `TEI` (17) ; des messages d'erreur sont également présents en annexes, en `JSON` (22), et, là encore, en `TEI` (21). Un exemple de réponse complète est également accessible depuis le dépôt en ligne de ce mémoire, puisque le document est trop long pour être inclus en annexes.

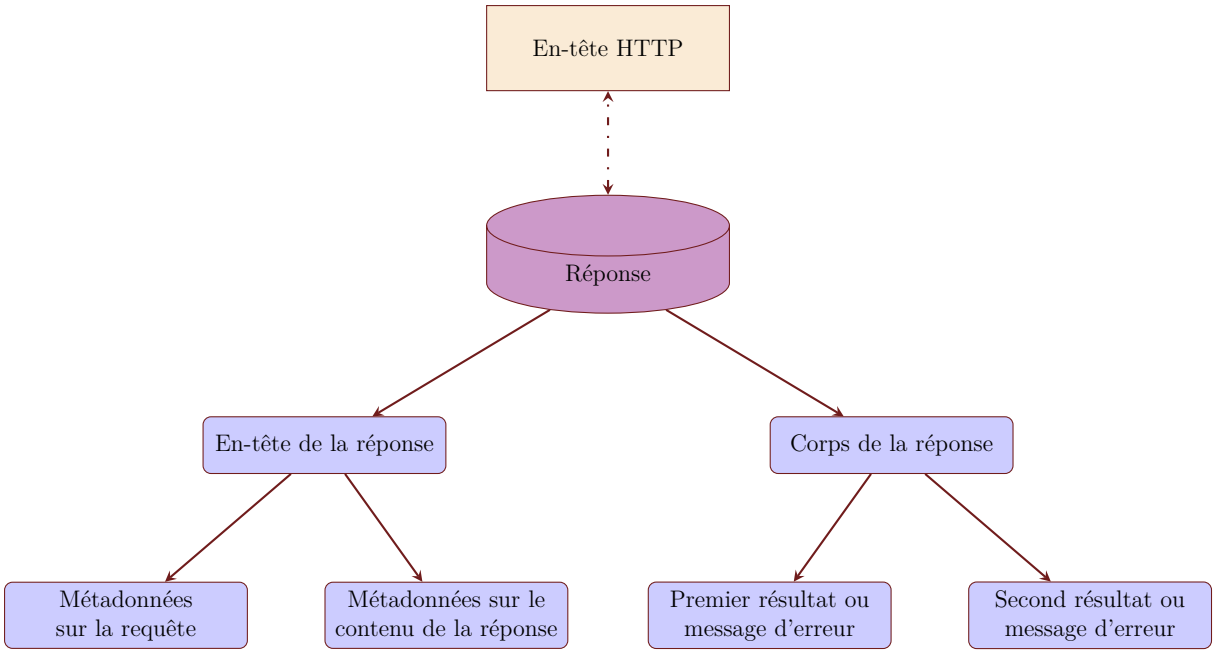


FIGURE 9.1 – Modèle de réponse renvoyé par *KatAPI*

message d'erreur descriptif. Cette séparation de la réponse en un en-tête et un corps est obligatoire lors de l'utilisation de la `TEI` ; son utilisation par le `JSON` est également inspirée par la structure des réponses `SPARQL` [4], bien que celles-ci contiennent beaucoup moins de métadonnées qu'une réponse de *KatAPI*.

9.4.2 L'en-tête des réponses, un conteneur pour les métadonnées

Si l'en-tête n'est pas voué à être conservé par les utilisateur.ice.s ayant réalisé des requêtes, il est néanmoins essentiel. Il permet à une personne qui n'est pas familière avec le projet d'aborder les données reçues et identifie précisément à quels critères elles correspondent. Il est garant du respect des principes FAIR, puisqu'il contient suffisamment de métadonnées pour que quiconque manipule la réponse dispose d'informations suffisantes à leur réutilisation. Décrivant le contexte d'une réponse, il rend la réponse auto-descriptive, ce qui correspond avec l'utilisation des principes REST. Les informations relatives à la requête sont les suivantes (9.1, 9.3) :

— La requête, représentée sous la forme d'une table associant les paramètres de recherche utilisés (`level, format...`) aux valeurs que l'utilisateur.ice leur a associées.

— La date à laquelle la requête est reçue, au format ISO. Cette information est surtout utile dans le cas où la réponse serait enregistrée par l'utilisateur.ice pour y accéder plus tard. Cela permet de dater les données produites, ce qui est utile puisque de

4. D. Beckett, J. Broekstra et S. Hawke, *SPARQL Query Results XML Format (Second Edition)*…

```
{
    "license": "Attribution 2.0 Generic (CC BY 2.0)",
    "query": {
        "format": "json",
        "level": "cat_stat",
        "name": "RDA",
        "sell_date": "1800-1900"
    },
    "query_date": "2022-08-24T16:41:42.604421",
    "status_code": 200
    # reste de l'en-tête
}
```

Code source 9.1 – Extrait d'en-tête de réponse en `JSON` décrivant le contexte d'une requête

nouveaux catalogues peuvent être ajoutés ultérieurement ; dans ce cas, la réponse du serveur ne serait plus à jour.

— Le statut HTTP de la réponse. Celui-ci est également présent dans l'en-tête HTTP, et sa présence ici peut donc sembler superflue. Il a été rajouté à cause du système de gestion des erreurs. Un choix de conception de l'application a en effet été de construire des messages d'erreurs sous la forme de `JSON` ou de `TEI` valides. Cela veut dire qu'une erreur retourne une réponse valide qui décrive l'erreur. Dans le cas où les réponses obtenues par le client sont traitées automatiquement, un programme informatique ne détectera donc pas forcément de problème. L'ajout d'un code de statut permet d'adapter facilement un programme à ce cas de figure, en vérifiant à chaque réponse le code HTTP dans le corps du document.

Les informations relatives aux données contenues dans le corps de la réponse, elles, permettent de mieux comprendre les données. Ces métadonnées sont différentes selon si le format de réponse est le `JSON` ou le `TEI`. Dans les deux cas, une réponse contient une licence, qui définit de quelle manière les données peuvent être utilisées. La licence Creative Commons Attribution 2.0 Generic (CC BY 2.0) est utilisée par le projet. Étant une licence libre, elle permet aux utilisateur.ice.s de réutiliser les données en les citant. Toutes les réponses partagent également une ou plusieurs « taxonomies » (l'usage de ce terme est issu de la `TEI`). Celles-ci servent à définir les termes spécifiques au projet, qui ne seraient pas nécessairement compréhensibles par tou.te.s les utilisateur.ice.s. Les taxonomies associent à des termes présents dans le corps de la réponse une définition. L'explication du terme `variance_price_c` dans une taxonomie en `TEI` est visible dans l'exemple 9.2. Ces taxonomies améliorent la compréhension de la réponse par une personne humaine ; mais elles offrent aussi des possibilités de traitement par une machine, puisqu'il est par exemple possible de remplacer un terme spécifique au projet par une définition.

Les taxonomies présentes dans une réponse changent selon le point d'accès aux données choisi, puisque ce sont des termes différents qui auront besoin d'être expliqués à chaque fois.

```
<category xml:id="cat_stat_keys_variance_price_c">
    <catDesc>The variance of the prices inside the catalogue.</catDesc>
</category>
```

Code source 9.2 – Un élément d'une taxonomie en TEI

Si une réponse est encodée en TEI, alors d'autres informations sont présentes. Deux raisons sont derrière ce choix, qui implique que les réponses en JSON et en TEI diffèrent. D'abord, la TEI définit toute une sémantique qui permet de caractériser très efficacement le contexte de production des données. Elle requiert qu'un document contienne un minimum d'informations dans son en-tête, ce qui a encouragé à étoffer l'en-tête. Ensuite, il a été considéré que la TEI est plus susceptible que le JSON d'être un format définitif, dans lequel la richesse d'informations privilégiée. Un JSON, au contraire, est utilisé parce qu'il est pratique, et il est susceptible d'être retraité par un.e utilisateur.ice de l'API afin qu'il s'adapte à ses besoins. Par ailleurs, la présence ou l'absence de ces données ne change pas la structure globale de la réponse, mais seulement l'en-tête. Les informations supplémentaires contenues dans le TEI sont notamment un titre (*KatAPI query results*, soit « Résultats de requête KatAPI ») et la mention des différents projets qui ont produit les données (*e-Ditiones, Katabase et Manuscript SaleS Catalogues*) (19).

Les métadonnées relatives à la requête doivent être encodées de façon légèrement différente lorsqu'un catalogue entier est requêté. Dans ce cas, la réponse n'est pas créée *ex nihilo*, elle est construite en récupérant un document entier dans la base de données de catalogues. Cela pose une question à propos de la nature du document : faut-il modifier le catalogue encodé, en sachant que celui-ci est déjà un objet édité et pensé de façon cohérente ? Le document transmis au client lors d'une requête est-il différent de celui qui est présent dans le serveur ? En plus de ces questions pratiques, le problème est technique : les métadonnées relatives à la requête sont normalement encodées en TEI dans un élément `tei:publicationStmt`, qui décrit le contexte de publication d'un document électronique. Dans un catalogue complet, cet élément est déjà utilisé. Le fait que cet élément soit déjà utilisé, mais qu'il doive être réutilisé lors d'une requête résume bien la question posée ci-dessus. La réponse à ce problème est pragmatique. Il est toujours possible que, à cause d'un problème interne à l'API ou dû au transfert de la réponse au client, que le catalogue soit corrompu ou contienne des erreurs. Dans ce cas, le catalogue transmis n'est pas différent de celui présent dans le serveur, mais il peut être considéré comme une édition de celui-ci. De ce fait, il est toujours nécessaire d'encoder le contexte dans lequel le catalogue a été transmis au client. La solution trouvée consiste à intégrer le contexte de la

requête à l'intérieur d'un élément `tei:availability` – qui définit les conditions d'accès à un document – dans le `tei:publicationStmt` (9.3). Cela permet de ne pas perdre d'informations de l'original, tout en intégrant à la réponse les métadonnées relatives à la requête.

```
<publicationStmt>
    <publisher>Projet e-Ditiones, Université de Neuchâtel</publisher>
    <availability status="restricted">
        <licence target="https://creativecommons.org/licenses/by/2.0">A
        ↪ ttribution 2.0 Generic (CC BY
        ↪ 2.0)</licence>
        <p>KatAPI query results. File created automatically by KatAPI,
        ↪ an API developped as part of the<ref
        ↪ target="https://katabase.huma-num.fr/">Manuscript SaleS
        ↪ Catalogues project</ref></p>
        <p>Query run on<date when-iso="2022-08-24T16:41:52.101590">2022
        ↪ -08-24T16:41:52.101590</date></p>
        <p>Query ran with HTTP status code:<ref target="https://develop
        ↪ er.mozilla.org/en-US/docs/Web/HTTP/Status/200"/></p>
        <p>The current file has been retrieved and updated as a
        ↪ response to the query:
            <table>
                <head>Query parameters</head>
                <row>
                    <cell role="key" xml:id="id">id</cell>
                    <cell role="key"
                    ↪ xml:id="output_format">format</cell>
                    <cell role="key" xml:id="level">level</cell>
                </row>
                <row>
                    <cell role="value" corresp="id">CAT_000300</cell>
                    <cell role="value" corresp="format">tei</cell>
                    <cell role="value" corresp="level">cat_full</cell>
                </row>
            </table>
        </p>
    </availability>
</publicationStmt>
```

Code source 9.3 – Exemple de `tei:publicationStmt` décrivant le contexte de la requête

L'en-tête sert à contextualiser les données contenues dans le corps de la réponse, afin qu'elles aient du sens au moment où elles sont reçues, mais aussi sur le long terme. En décrivant le contexte de la requête et de la production des docuemnt, et en indiquant la licence sous laquelle les documents sont diffusés, l'en-tête cherche à s'adapter aux principes FAIR. L'éditorialisation de l'en-tête cherche également à construire une structure

commune pour les différents points d'entrée, en accord avec les principes REST. Son rôle principal est de permettre la compréhension du corps des réponses, qui sont elles mêmes issues d'un travail d'éditorialisation.

9.4.3 Les corps de réponse

À un niveau abstrait, les corps de réponse prennent toujours la forme d'une liste contenant les résultats pertinents en fonction de la requête faite par l'utilisateur.ice. Selon le point d'accès aux données, les résultats obtenus diffèrent.

Le corps d'une réponse au format `cat_full`

Ce cas particulier implique un fonctionnement particulier de l'API : plutôt que de construire un jeu de données sur mesure à transmettre au client, il s'agit de vérifier si le fichier demandé existe. Cela implique, comme on l'a vu, un en-tête de réponse est particulier ; le corps l'est aussi. Dans le cas où une catalogue est trouvé, c'est le corps de ce catalogue qui est retourné au client. Cependant, si aucun catalogue correspondant à l'identifiant fourni par le client n'est trouvé, le corps comme l'en-tête sont différents. Il n'est pas possible de reprendre un en-tête existant, et un en-tête sur mesure est donc constitué, comme pour les points d'entrée `cat_stat` et `item`. Le corps est vide, puisqu'aucun résultat n'a été trouvé (9.4).

```
<TEI>
    <teiHeader>
        <!-- ... -->
    </teiHeader>
    <text>
        <body>
            <div type="search-results"/>
        </body>
    </text>
</TEI>
```

Code source 9.4 – Extrait de réponse au niveau `cat_full` lorsqu'aucun catalogue n'est trouvé

Le corps d'une réponse au format `cat_stat`

À ce niveau de granularité, la réponse consiste en une liste de résultats correspondants aux paramètres de recherche fournis par l'utilisateur.ice. Chaque résultat est un identifiant de catalogue associé à des statistiques sur celui-ci ; si les manuscrits vendus dans le catalogue sont à prix fixe (par opposition aux items vendus aux enchères), des

statistiques supplémentaires sont incluses sur les prix. Il est considéré que l'intérêt de ce niveau de granularité est de permettre une comparaison entre les différents catalogues ; c'est pourquoi tous les prix sont exprimés en francs constants 1900. La liste complète des données pouvant être transmises au client sur un catalogue est donc :

— Le titre du catalogue.

— Le type de catalogue (*Revue des autographes, catalogue de vente aux enchères...*).

— Le nombre de manuscrits en vente dans un catalogue.

— L'année de la vente.

— La somme du prix de tous les items du catalogue.

— Le prix de l'item le plus cher en vente.

— La liste des items les plus cher du catalogue.

— Le prix le plus faible pour un item.

— La moyenne, le premier quartile et la médiane du prix d'un item dans ce catalogue.

— Le mode du prix, c'est-à-dire le prix le plus fréquemment attribué à un item.

— La variance des prix des items, ce qui permet de mesurer et de comparer la volatilité des prix d'un catalogue à un autre.

Comment représenter ces données ? La représentation en `JSON` est assez simple, puisqu'il s'agit simplement de créer un dictionnaire associant des clés (identifiant le type de donnée) à l'information elle-même (9.5 [5]). Il est cependant plus difficile d'exprimer ces résultats dans un fichier `TEI`. Celle-ci impose des contraintes strictes sur la manière de structurer des éléments : seulement certains d'entre eux sont autorisés à certains endroits, et certains éléments doivent obligatoirement en contenir d'autres, par exemple. De plus, chaque élément a sa signification propre, qui doit être respectée dans la construction de la réponse ; la `TEI` étant pensée pour le texte, il n'est pas évident de l'adapter à une liste de statistiques. Mais le véritable problème n'est pas seulement technique : la `TEI` est, par définition, spécifique et propre à un projet. Il existe une certaine liberté dans l'utilisation des éléments, de sorte que différents encodages soient des représentations différentes d'un même texte ; un encodage est donc une interprétation du texte, qui dépend des visées d'un projet. À l'inverse, le format de réponse d'une API doit être générique et pratique : être facilement utilisable et compréhensible (même sans connaissance approfondie de la `TEI`) et pouvoir s'adapter aux différents besoins des utilisateur.ice.s, sans privilégier une vision scientifique. Par expérience, les formats de réponse retournés par une API ne correspondent en général pas aux besoins de l'utilisateur.ice de l'API, qui récupère les données dans les réponses pour les adapter à ses propres besoins ; dans ce cas, il est nécessaire de manipuler le `XML` retourné et, par expérience, plus ce format est complexe, plus il est

5. Un exemple de réponse complet est disponible en annexes (20).

difficile de le manipuler avec des méthodes computationnelles. Une arborescence la plus simple possible doit donc être privilégiée, afin de faciliter sa manipulation. Cela encourage la création d'un format le plus générique possible. Faire coïncider un besoin de généricité avec un format qui encourage la spécificité n'est pas évident.

```
"CAT_000018": {
    "cat_type": "RDA",
    "currency": "FRF",
    "high_price_c": 561.0,
    "high_price_items_c": {
        "CAT_000018_e144": 561.0
    },
    "item_count": 192,
    "low_price_c": 1.53,
    "mean_price_c": 10.565811518324606,
    "median_price_c": 5.1,
    "mode_price_c": 3.06,
    "sell_date": "1873-08",
    "title": "RDA, N\u00b037 (ao\u00fbt 1873)",
    "total_price_c": 2018,
    "variance_price_c": 1658.1071772210191
}
```

Code source 9.5 – Représentation `JSON` des données portant sur un catalogue

Aussi le parti pris a-t-il été d'encoder les données de la façon la plus simple possible (9.6 [6]). Les différents catalogues sont contenus dans un `tei:list` ; en s'inspirant de l'encodage choisi pour les manuscrits dans les catalogues, chaque catalogue est représenté dans un `tei:item`. À l'intérieur est contenu un `tei:label` qui contient l'identifiant du catalogue. Les différentes données sont contenues dans un `tei:term`, un élément qui permet d'encoder un terme ou un symbole technique [7]. Ce choix est adapté, puisque les différents termes associés aux données sont propres au projet. Le terme en lui-même est encodé dans un attribut `@key`, tandis que la valeur est dans le corps de l'élément. Cela permet de reproduire l'association clé-valeur utilisée dans le `JSON` afin de permettre une similarité entre les deux formats de représentation des données. Pour maintenir l'arborescence la plus simple possible, la liste des items les plus chers est contenue à la racine du `tei:item` plutôt que dans une liste imbriquée. Dans ce cas, le prix se trouve dans le corps de l'élément, et l'identifiant du manuscrit dans un attribut `@ana`. L'usage d'attributs permet de caractériser les éléments de façon bien plus précise qu'avec un `JSON`. Un attribut `@type` est utilisé pour caractériser certains types de données (date, prix constants) ; lorsqu'une requête est faite en utilisant certains paramètres, comme `sell_date`, et que ces para-

6. Pour un exemple de réponse complète, voir en annexes (19).
7. TEI Consortium, *P5 : Guidelines for Electronic Text Encoding and Interchange…*

mètres sont présents dans la réponse, alors un attribut `@ref` est utilisé pour pointer vers le descriptif de la requête dans l'en-tête. Cela permet de lier corps et en-tête des données.

```
<item ana="CAT_000362">
    <label>CAT_000362</label>
    <term key="title">Vente Jacques Charavay, août 1875, nº 185</term>
    <term key="cat_type">LAC</term>
    <term key="sell_date" type="date">1875</term>
    <term key="item_count">106</term>
    <term key="currency">FRF</term>
    <term key="total_price_c" type="constant-price">1039</term>
    <term key="low_price_c" type="constant-price">1.27</term>
    <term key="high_price_c" type="constant-price">102.0</term>
    <term key="mean_price_c"
    ↪ type="constant-price">9.810188679245282</term>
    <term key="median_price_c" type="constant-price">4.59</term>
    <term key="mode_price_c" type="constant-price">3.06</term>
    <term key="variance_price_c"
    ↪ type="constant-price">194.2879773228907</term>
    <term key="high_price_items_c" type="constant-price"
    ↪ ana="CAT_000362_e27096">102.0</term>
</item>
```

Code source 9.6 – Représentation en `XML-TEI` des données portant sur un catalogue

Ces choix d'encodage permettent de construire des réponses simples, avec un minimum d'imbrication entre les éléments ; l'utilisation d'attributs permet de créer des liens entre mots clés et valeurs, mais aussi des liens entre différentes parties de la réponse (puisque l'attribut `@key` du corps de la réponse est défini dans l'élement `tei:taxonomy` de l'en-tête). Un format de réponse en `TEI` permet donc de commenter et de caractériser les données avec une précision qui n'est pas possible dans un `JSON`. Cependant, grâce aux choix de modélisation, une représentation des données en `TEI` présente la même structure qu'un `JSON`, soit une série de couples clés-valeurs.

Le corps d'une réponse au format `item`

La problématique de la modélisation des réponses au niveau de l'item de catalogue est bien différente de ce qui a été exposé pour le niveau `cat_stat`. Pour le point d'accès présenté ci-dessus, les données en `TEI` sont créées *ex nihilo* à partir d'un `JSON`. Il était donc nécessaire de trouver un moyen de les représenter dans un autre format. Une autre spécificité du point d'accès présenté ci-dessus est qu'il ne contient que des statistiques ; de fait, la `TEI` n'est pas pensée pour ce type de format, ce qui demande de trouver un moyen pour adapter ces données dans ce format. Au niveau de l'entrée de catalogue, les données représentées sont des textuelles, puisque la réponse envoyée par le serveur contient une

liste d'entrées de catalogue. De plus, ces données ont déjà été encodées en `TEI`, dans les catalogues originels. Le choix a été de ne pas modifier ces représentations : elles sont le fruit d'un travail scientifique d'édition, d'extraction de données, de normalisation et d'enrichissement. Les catalogues, encodés en `TEI`, ne sont donc pas seulement représentatifs des choix au moment de la première édition numérique des catalogues. Ils contiennent également des informations produites tout au long de la chaîne de traitement, dont les identifiants *Wikidata* extraits lors de la résolution d'entités nommées décrite dans la partie précédente. L'encodage des items, tel qu'il est fait dans les catalogues, est donc une représentation synthétique de ceux-ci, qui témoigne des choix scientifiques faits tout au long de la chaîne de traitement. Conserver cet encodage permet de disposer d'une représentation complète et déjà éditorialisée des données ; cela permet également de représenter tout le travail fait sur les données, et donc de diffuser celles-ci de manière transparente, en accord avec les principes FAIR. Les choix d'encodage pour les sont également précisément documentés, ce dont peut bénéficier un.e chercheur.euse souhaitant les réutiliser. Par ailleurs, les données diffusées par *KatAPI* ont traversé toute une chaîne de traitement ; celle-ci est susceptible de s'étoffer, avec l'ajout de nouvelles étapes ; dans ce cas, l'encodage des catalogues évoluerait lui-aussi. Il est alors naturel que la représentation de ces données évolue elle aussi, afin de s'adapter aux évolutions scientifiques du projet. Pour toutes ces raisons, il est plus intéressant de reprendre l'encodage déjà réalisé plutôt que de recréer une nouvelle représentation. Le problème est que les formats de réponse pourront alors changer, ce qui peut être peu pratique pour les utilisateur.ice.s de l'API (qui, si il.elle.s retraitent les réponses automatiquement, devront modifier leur chaîne de traitement). Cependant, ces évolutions sont justifiées et souhaitables, puisqu'elles reflètent l'évolution d'un travail scientifique. Le corps des réponses, en `TEI` et au niveau `item`, est donc composé d'une `tei:list` contenant toutes les entrées de catalogues représentées par un `tei:item` (9.7 [8]).

La représentation en `JSON` des entrées de catalogue existait elle aussi avant le développement de l'API. Cette représentation cherchant à être l'équivalent le plus fidèle possible des entrées de catalogues encodées en `TEI`, elle a été réutilisée pour les réponses de l'API (9.8 [9]). Au niveau `item`, la représentation des données est donc équivalente à leur représentation dans les différentes bases de données utilisées par l'API. Cette représentation implique nécessairement une perte d'information. En premier lieu, seul le nom de famille de l'auteur.ice est conservé ; l'élément `tei:note` est supprimé, de même que le numéro de lot dans le catalogue. La représentation en `JSON` contient également toutes les informations produites durant la chaîne de traitement (normalisation, enrichissement), puisque ces informations, contenues dans des attributs, ont été extraites. Comme pour les réponses en `TEI`, les termes normalisés et spécifiques au projet sont définis au sein de taxonomies dans l'en-tête. Au premier abord, utiliser la structure de données présente dans les

8. Pour un exemple de réponse complète, voir l'annexe 17.
9. Pour un exemple de réponse complète de l'API en `XML-TEI` au niveau `item`, voir l'annexe 18.

```xml
<list>
    <head>Search results</head>
    <item n="108" xml:id="CAT_000204_e108">
        <num type="lot">108</num>
        <name type="author" ref="wd:Q255">BEETHOVEN (L. van)</name>
        <trait>
        <p>le grand compositeur de musique.</p>
        </trait>
        <desc xml:id="CAT_000204_e108_d1">
        <term ana="#document_type_9">L. s.</term> à M. M. Schlesinger, à
        ↪ Berlin; Vienne, <date when="1820-05-31">31 mai 1820</date>,
        ↪ <measure type="length" unit="p" n="2">2 p.</measure>
        <measure type="format" unit="f"
        ↪ ana="#document_format_4">in-4</measure>, cachet</desc>
        <note>Curieuse lettre sur ses ouvrages. Il leur accorde le droit
        ↪ de vendre ses compositions en Angleterre, y compris les airs
        ↪ écossais, aux conditions indiquées par lui. Il s'engage à
        ↪ leur livrer dans trois mois trois sonates pour le prix de 90
        ↪ florins qu'ils ont fixé. C'est pour leur être agréable qu'il
        ↪ accepte un si petit honoraire. « Je suis habitué à faire des
        ↪ sacrifices, la composition de mes OEuvres n'étant pas faite
        ↪ seulement au point de vue du rapport des honoraires, mais
        ↪ surtout dans l'intention d'en tirer quelque chose de bon
        ↪ pour l'art.»</note>
    </item>
    <!-- autres items --->
</list>
```

Code source 9.7 – Représentation en `XML-TEI` des réponses de l'API au niveau `item`

bases de données pour construire une réponse peut sembler aller à l'encontre du principe d'uniformité de l'interface du REST : la structure des réponses n'est pas indépendante de la source de données. Ce choix se justifie cependant par les spécificités des données de recherche. Les informations, telles qu'elles sont stockées dans les différents fichiers consultés par l'API, sont représentatifs de ce travail de recherche et de choix d'encodage spécifiques. Réutiliser ces structures de données permet de transmettre non seulement les données, mais l'intégralité de la recherche, puisque l'encodage numérique fait partie intégrante du travail scientifique. La structure des données est par ailleurs relativement stable sur le long terme ; en évoluant, elle ne change pas drastiquement, puisque seulement de nouveaux attributs ou couples clés-valeurs sont ajoutés aux données. Cela permet donc une relative pérennité des représentations et des formats de réponse renvoyés au client. De plus, utiliser des données pré-existentes en `JSON` et en `TEI` garantit que, à ce niveau, la des données représentation a une structure équivalente dans les deux formats.

```
"CAT_000185_e608_d1": {
    "author": "S\u00e9vign\u00e9",
    "author_wikidata_id": null,
    "date": "1685",
    "desc": "\n                        L. aut. \u00e0 sa fille; mercredi
    ↪  14 f\u00e9vrier (1685), 7 p. 1/2\n                        in-4\n
    ↪  ",
    "format": 4,
    "number_of_pages": 7.5,
    "price": null,
    "sell_date": "1881",
    "term": 8
}
```

Code source 9.8 – Représentation en `JSON` des réponses de l'API au niveau `item`

Le corps d'une réponse en cas d'erreur

Comme cela a été dit, l'un des prérequis pour *KatAPI* était la création de messages d'erreurs structurés et autodescriptifs, en `TEI` ou en `JSON`. Deux types d'erreurs ont été définies. D'abord, les erreurs dues à une requête incorrecte de la part du client (soit qu'elle ne respecte pas la sémantique pour les requêtes, soit qu'elle utilise des valeurs non-autorisées). Ensuite, les erreurs de la part du serveur : l'application a été testée pour minimiser le risque d'erreurs, mais celles-ci sont toujours possibles ; un système de gestion des erreurs a donc été mis en place pour informer l'utilisateur.ice si une telle erreur a lieu. Les réponses retournées ont la même structure dans les deux cas. Les choix de représentation des erreurs ont été faits pour être conformes avec la structure globale définie pour les réponses (9.1), mais aussi pour être cohérents avec les choix d'encodages faits pour les autres représentations présentées ci-dessus. Les réponses sont donc représentées sous la forme d'une liste qui associe à des clés des descriptions des erreurs. Ces clés correspondent, lorsque cela est possible, au nom des paramètres pour lesquelles des erreurs ont survenu. Les messages permettent à l'utilisateur.ice de corriger son erreur afin de pouvoir récupérer les résultats. En plus de cette liste, un message introductif définit l'erreur qui a eu lieu afin de savoir si l'erreur est due au client ou au serveur ; pour aller dans le même sens, le statut HTTP est modifié dans l'entrée afin de caractériser précisément l'erreur. Lorsque le message d'erreur est en `JSON`, l'encodage de la réponse est relativement simple (9.9 [10]).

En `XML-TEI`, la structure des données est inspirée de celle des autres formats de réponse, et particulièrement des réponses obtenues au niveau `item` (9.10 [11]). Les différents messages d'erreur sont contenus au sein d'une liste ; chaque erreur correspond à un `tei:item` ; le code du message d'erreur se trouve dans un attribut `@ana` de cet élément et

10. Un message d'erreur complet de l'API est visible en annexes (22).
11. Un message d'erreur complet en `JSON` est visible en annexes : 21.

```
"results": {
    "__error_type__": "Invalid parameters or parameters combination",
    "error_description": {
        "format": "The format must match: (tei|json)",
        "id_incompatible_params": "Invalid parameters with parameter id:
        ↪  ['sell_date']",
        "name+id": "You cannot provide both a name and an id",
        "sell_date": "The format must match: \d{4}(-\d{4})?",
        "unallowed_params": "Unallowed parameters for the API: ['api']"
    }
}
```

Code source 9.9 – Corps de réponse en cas d'erreur de la part du client en `JSON`

dans le `tei:label`, ce qui permet d'identifier l'erreur ; le message d'erreur en lui-même est contenu dans un `tei:desc`. Ainsi, les messages d'erreur correspondent aux principes définis pour les réponses REST : ce sont des représentations qui sont toujours uniformes. Leur structure est de plus analogue aux autres représentations définies pour l'API, ce qui permet à un.e utilisateur.ice de comprendre plus facilement les différents formats de réponse. Enfin, ces messages d'erreurs ont l'avantage d'être entièrement auto-descriptifs, et de pointer vers une manière de corriger sa requête. Cela limite le besoin d'avoir à se référer à une documentation externe pour corriger ses erreurs. Les messages d'erreurs émis par l'API cherchent à être le plus facilement compréhensibles, autant du point de vue des messages eux-mêmes que de leur encodage. En effet, la structure définie pour ces réponses est la même que celle établie pour les réponses valides ; ce sont également les mêmes éléments de la `TEI` qui sont utilisés, ce qui permet qu'un.e utilisateur.ice ne soit pas dérouté.e par la structure du message d'erreur et puisse uniquement se concentrer sur son contenu.

Plus largement, les formats de réponses présentés dans cette partie sont garants à la fois du respect des principes REST. Ils sont tous des implémentations différentes d'un modèle commun, lui-même inspiré des structures en en-tête/corps utilisés par le format de réponse `SPARQL`, mais aussi par de nombreux standards d'encodage numérique, dont la `TEI`. Cette double inspiration reflète le statut intermédiaire des réponses envoyées par l'API au client : ce sont à la fois des réponses d'API (et donc des formats techniques, faits pour être manipulés par des machines) et des éditions crées à la demande pour répondre aux besoins d'utilisateur.ice.s. Répondre à leurs demandes implique également de respecter les principes FAIR, et c'est aussi à cette fin que les réponses de l'API ont été modélisées. En effet, celles-ci visent à contenir toutes les données et métadonnées nécessaires à la compréhension d'un résultat par une personne humaine ainsi qu'à son traitement par une machine. La quantité d'informations intégrées aux réponses permet de faire de celles-ci de simple formats de réponse qui seront modifiés par le client ; mais elles peuvent également

```
div type="error-message">
    <list>
        <head>Invalid parameters or parameters combination</head>
        <item ana="no_name+id">
            <label>no_name+id</label>
            <desc>You must specify at least a name or an id</desc>
        </item>
        <item ana="sell_date" corresp="sell_date">
            <label>sell_date</label>
            <desc>The format must match: \d{4}(-\d{4})?</desc>
        </item>
    </list>
</div>
```

Code source 9.10 – Corps de réponse en cas d'erreur de la part du client en `XML-TEI`

être conservées et restées intelligibles sur le long terme. La modélisation de l'API au croisement des besoins de chercheur.euse.s et de bonnes pratiques informatiques met en avant que ces deux besoins ne sont pas contradictoires. En encourageant à produire des données compréhensibles et réutilisables, le FAIR comme le REST encouragent à mieux penser la conception d'applications, afin de produire des outils utiles et des réponses qui soient détaillées sans être excessivement complexes.

Chapitre 10

Implémentation et fonctionnement interne de *KatAPI*

Le chapitre précédent s'attache à décrire le fonctionnement de l'API côté client, c'est-à-dire la manière dont ce dernier interagit avec l'application, en présentant notamment la sémantique des requêtes et la structure des réponses. Cependant, une API n'est pas seulement une interaction avec un client, c'est aussi un programme composé d'une série de fonctions chargées de traiter la requête du client, d'interagir avec la base de données et de construire une réponse. Ce chapitre décrit donc le fonctionnement de l'API côté serveur : comment les données sont reçues, la manière dont elles sont traitées et comment les réponses sont construites.

En suivant le principe de séparation des préoccupations, le fonctionnement interne de l'API (10.1) peut être divisé en trois parties : la réception des requêtes et la vérification de leur validité, l'interaction avec les bases de données et la construction des réponses.

10.1 Réception des requêtes

Comme cela a été précisé au début du chapitre précédent, l'API procède à un contrôle strict des requêtes afin de ne pas faire essayer de traiter des demandes contradictoires et de ne pas lancer de requêtes inutiles. Trois étapes de vérifications ont lieu afin de garantir le respect de la sémantique des requêtes :

— Validité des paramètres utilisés : l'API vérifie que la requête ne contienne pas de paramètres non autorisés.

— Validité des valeurs utilisés : l'API vérifie que toutes les valeurs associées à des paramètres soient autorisées. Cette validation est faite à l'aide d'expressions régulières pour les champs qui demandent des informations normalisées, tel que les dates (qui doivent être au format « AAAA » ou « AAAA-AAAA »).

— Absence de paramètres contradictoires : certains paramètres ne sont pas autorisés avec d'autres, car cela entraînerait la construction de requêtes incohérentes, en cherchant par exemple un nom d'auteur.ice et un identifiant qui ne correspondent pas. Pour éviter ce cas de figure, l'API vérifie qu'une requête ne contienne pas de combinaisons « non-autorisées ».

Si la vérification se fait en plusieurs étapes, elle a lieu intégralement pour chaque requête (c'est-à-dire que l'exécution de la vérification ne cesse pas à la première erreur rencontrée). Cela permet d'obtenir une liste complète de toutes les erreurs présentes dans la requêtes : à chaque erreur détectée par l'API est associé un mot clé. Si la requête n'est pas valide, alors il n'y a pas d'interaction avec le serveur. Une erreur HTTP 422 *Unprocessable Content* (*Contenu ne pouvant être traité*) est faite. Ce code d'erreur signifie que la requête est correcte d'après le protocole HTTP (sans quoi elle ne serait pas parvenue au serveur), mais qu'elle ne peut pas être traitée par le serveur [1]. Le client doit donc modifier sa requête avant de la relancer. La liste d'erreurs constituée pendant la vérification est alors passée à un constructeur de réponse au format défini par le client (`JSON` ou `TEI`). Ce constructeur de ajoute à la réponse un message pour chaque erreur rencontrée ; il définit également l'en-tête de la réponse, avec le contexte de la requête, des taxonomies (éventuellement) et enfin le statut HTTP. Pour finir, les en-têtes HTTP sont établis, avec notamment le statut de la réponse et son type MIME, qui garantissent que le client pourra traiter ce que le serveur lui envoie. L'interaction avec le serveur prend alors fin.

10.2 Interaction avec les bases de données et construction des réponses

Si aucune erreur n'a été identifiée, alors la requête est traitée par le serveur. Dans un premier temps sont définies des valeurs par défaut pour les requêtes. Si aucune valeur n'a été fournie pour le paramètre `format`, alors le format de réponse est le `JSON` ; si aucun point d'accès n'a été défini, alors des données sont traitées au niveau du manuscrit. Commence alors la phase de récupération des données. Celle-ci est très différente en fonction du point d'accès défini par l'utilisateur.ice.

10.2.1 Au niveau du catalogue complet (`cat_full`)

C'est au niveau du catalogue complet que le traitement est le plus simple : l'API se contente de chercher si le catalogue demandé par l'utilisateur.ice existe. Si c'est le

1. R. Fielding, Mark Nottingham et Jilian Reschke, *HTTP Semantics. Request for Comments : 9110*, HTTP, juin 2022, URL : `https://httpwg.org/specs/rfc9110.html`, §15.5.21. *422 Unprocessable Content*.

cas, alors ce catalogue est passé à un constructeur de réponse. Celui-ci complète l'en-tête du catalogue en ajoutant dans le `tei:publicationStmt` le contexte de la requête (date, paramètres de la requête) et le statut HTTP 200, qui indique que l'interaction s'est déroulée normalement [2]. Si le catalogue demandé n'est pas trouvé, alors le constructeur de réponse adopte un fonctionnement différent. Normalement, la réponse pour ce point d'accès est construite en ajoutant des données à un document pré-existant. Si celui-ci n'existe pas, alors une réponse complète doit être créée *ex-nihilo* pour être renvoyée à l'utilisateur.ice. Dans ce cas, la réponse créée prend le même format que celle des deux autres points d'accès. Le constructeur renseigne le contexte de la requête et ajoute le statut HTTP 200. Celui-ci est utilisé même lorsqu'aucune réponse n'est obtenue : le fait que la ressource recherchée par le client n'existe pas ne veut pas dire pour autant qu'il y ait eu une erreur. Enfin, les en-têtes sont construits pour que la réponse soit renvoyée au client.

10.2.2 Aux niveaux des manuscrits et des statistiques sur les catalogues (`item` et `cat_stat`)

Aux niveaux `cat_stat` et `item`, le traitement d'une requête fonctionne de façon analogue. Deux bases de données en `JSON` contiennent des données sur tous les manuscrits et tous les catalogues. L'application commence alors par chercher dans ces fichiers tous les catalogues ou manuscrits pertinents. La recherche se fait en deux temps.

D'abord, l'application identifie toutes les manuscrits dont l'auteur.ice correspond au nom fourni par le client, ou tous les catalogues dont le type correspond à celui recherché par le client (*RDA, AUC...*). Pour augmenter le nombre de résultats pertinents à retourner à l'utilisateur.ice, la recherche se fait en simplifiant à la fois la requête et les informations présentes dans la source de données. Par exemple, quand un manuscrit est recherché à partir du nom d'un.e auteur.ice, cette recherche se fait en comparant le nom fourni par le client avec celui présent dans la source de données. Normalement, cette recherche se fait par alignement complet des informations de la requête avec celles du fichier `JSON`. Les données sont donc retraitées pour que la comparaison se fasse sans prendre en compte les différences de casse, d'accents et de ponctuation. Par exemple, si le client recherche tous les manuscrits écrits par « sévigné », alors des manuscrits dont l'autrice est décrite comme étant « SÉVIGNÉ », « Sévigné » ou « SEVIGNE » seront également jugés pertinents. Cette simplification de chaînes de caractères va dans le même sens que les méthodes utilisées par les moteurs de recherche contemporains. Cependant, la recherche pourrait être alignée plus finement encore, par exemple en sélectionnant les items pertinents non pas par correspondance de chaînes de caractères, mais en calculant une distance de Levenshtein. Celle-ci correspond au nombre de caractères de différence entre deux chaînes de caractères,

2. *Ibid.*, §15.3.1 200 *OK*.

et permet donc d'inclure un certain degré de bruit qui peut être très utile pour récupérer tous les résultats pertinents, surtout au vu des erreurs d'OCR qui rajoutent du bruit aux données. La recherche par nom d'auteur.ice.s a également une faiblesse majeure : elle ne se fait que par noms de famille, puisque ce n'est que ceux-ci qui sont présents dans la source de données.

Ensuite, ces résultats sont filtrés en fonction des autres paramètres de la requête du client, à savoir les dates de vente et, pour les manuscrits, de création. Si une date fournie correspond à une année, alors l'application ne conserve que les entrées datant de cette année. Si c'est au contraire une plage de dates qui est fournie, alors l'application ne conserve que les entrées qui sont contenues dans cette plage. Pour les manuscrits, date de création et date de vente peuvent être fournies. Dans ce cas, un manuscrit doit correspondre à ces deux dates (ou plages de dates) pour être conservé.

Une fois les données pertinentes extraites, soit une réponse est construite en `JSON`, soit ces résultats sont transposés en `TEI`. C'est au niveau de la transposition qu'apparaît la différence de traitement entre les deux niveaux. Au niveau `cat_stat`, il n'existe pas de représentation `TEI` des statistiques sur les différents catalogues. L'ensemble de ces statistiques sont donc extraites de la base de données en `JSON` avant d'être traduites en un `TEI`. Lorsque des informations sont recherchées au niveau des manuscrits, cependant, une représentation `TEI` existe déjà. C'est donc elle qui doit être renvoyée à l'utilisateur.ice. Pourquoi, alors, ne pas rechercher directement les données pertinentes dans les fichiers `TEI` ? Ce choix s'explique par des raisons de performance. Si les données sont directement extraites de la `TEI`, alors l'application doit rechercher des manuscrits dans près de 500 fichiers, ce qui demande d'ouvrir tous ces fichiers, de lire leur contenu et de rechercher à l'intérieur les manuscrits correspondants. Au contraire, la base de données sur les manuscrits en `JSON` n'est qu'un seul (très volumineux) fichier. Il est donc plus rapide de commencer par ouvrir cette source de données pour extraire tous les identifiants des manuscrits correspondant aux paramètres souhaités. Un identifiant de manuscrit commençant par l'identifiant du catalogue dans lequel il est contenu, il ne reste plus qu'à ouvrir les catalogues pertinents pour récupérer les descriptions des manuscrits encodées en `TEI`.

Pour finir, des réponses complètes sont construites à partir des données extraites. En `JSON` comme en `TEI`, la méthode est la même que celle qui a déjà été présentée : l'en-tête du document est reconstruite pour contenir les métadonnées décrivant la requête et les données contenues dans le corps de la réponse ; ensuite, l'en-tête HTTP est défini, et la réponse est retournée à l'utilisateur.ice.

10.3 Gestion des erreurs imprévues

Le processus de traitement d'une requête a été testé, et ne rencontre théoriquement pas d'erreur. Cependant, l'application est exposée au public ; une erreur imprévue est donc toujours possible, du fait d'une requête du client qui comprenne un problème imprévu ou d'une erreur qui n'ait pas été rencontrée pendant la conception de l'application et la rédaction des tests. Comme il n'est pas possible que l'application s'arrête tout simplement de fonctionner en cas d'erreur imprévue, un processus de gestion des erreurs internes a été défini. Si une erreur arrive entre le moment où une requête commence à être traitée et le moment où l'interaction prend fin, alors une erreur HTTP *Internal Server Error* (« Erreur de serveur interne ») 500 est émise. Ce code d'erreur est un code générique qui indique au client qu'une erreur imprévue a été rencontrée [3]. L'émission de cette erreur lance un processus isolé du reste de l'application de construction et de renvoi d'un message d'erreur. Ce processus étant entièrement séparé de la chaîne de traitement « normale » d'une requête, il ne risque pas d'être impacté par une autre erreur issue de *KatAPI*. En cas d'erreur 500, une réponse est construire en `TEI` ou en `JSON`. Elle contient dans son en-tête le contexte de la requête et la taxonomies nécessaires ; le corps de la réponse est fait d'un message indiquant qu'une erreur imprévue a été rencontrée. Comme toujours avant l'envoi d'une réponse au client, les en-têtes HTTP sont définis ; enfin, la requête est envoyée au client.

3. *Ibid.*, §15.6.1. 500 *Internal Server Error*.

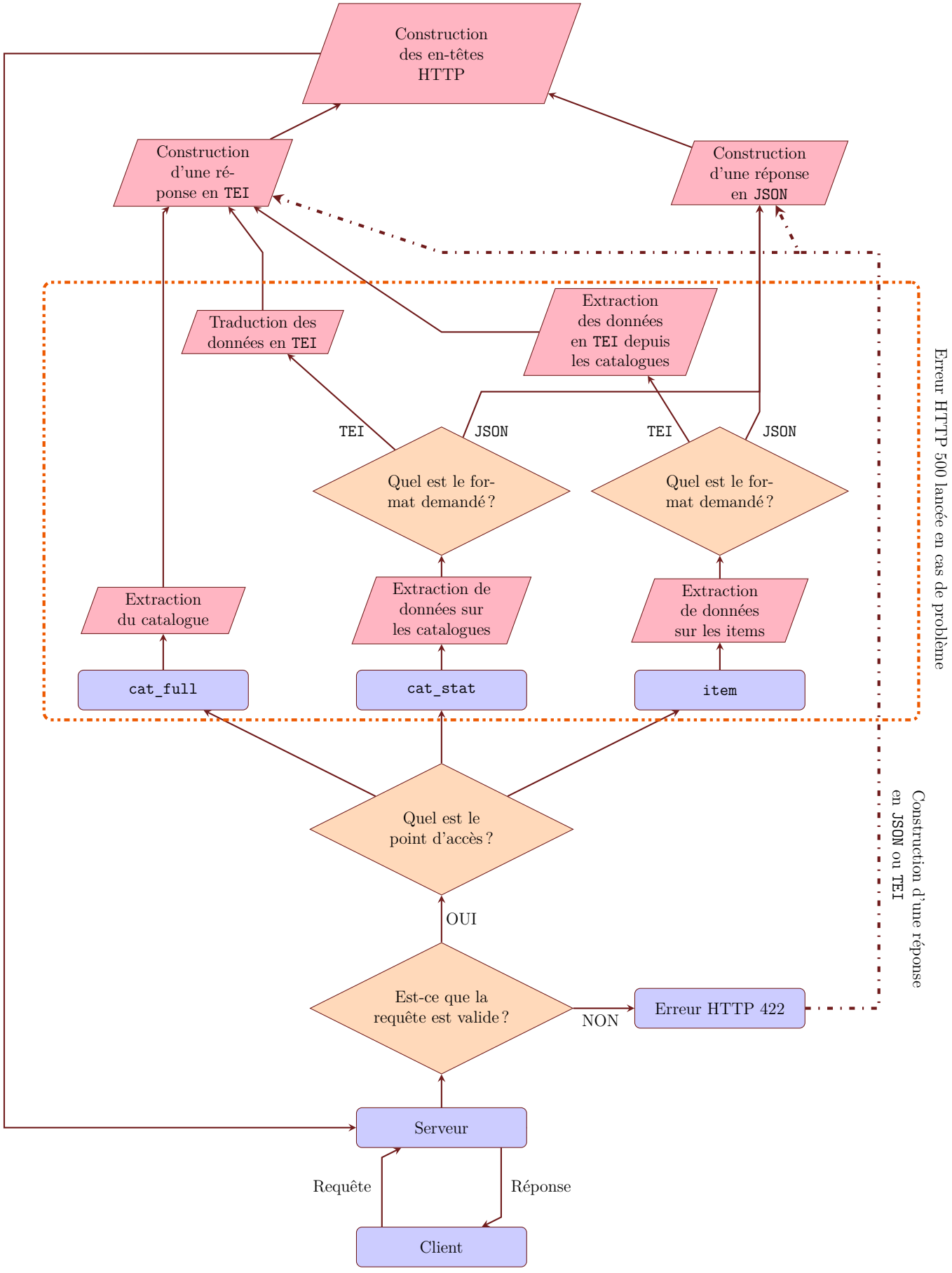


FIGURE 10.1 – Fonctionnement interne de *KatAPI*

10.4 En conclusion

L'application présentée ici vise à être un outil pour chercheur.euse.s, qui permette le partage de données de façon automatisée. Cela peut faciliter la récupération de données brutes, sans avoir à passer par un téléchargement « à la main ». L'objectif est donc d'enrourager la réutilisation de données dans un contexte de science ouverte. L'API va cependant plus loin que la simple récupération de données brutes, puisqu'elle fonctionne comme un moteur de recherche à facettes. Elle permet par exemple de rechercher tous les manuscrits vendus entre 1880 et 1890 et écrits par Mme de Sévigné entre 1660 et 1680. Cette fonctionnalité, et la possibilité de mener des recherches à différents niveaux, peut aider à la constitution de jeux de données sur mesure, ce qui peut être utile autant dans la suite du projet que pour d'autres chercheur.euse.s. Si les données partagées sont brutes, leur partage demande d'être tout sauf brut. Des données hors contexte sont inutilisables, autant par une personne humaine que par une machine. C'est pourquoi deux standards ont été suivis : l'un technique (le REST) et l'autre scientifique, le FAIR. Ces standards et principes servent à la recherche, puisqu'ils permettent tout simplement la production de données utilisables par des chercheur.euse.s. Cependant, ils aident également à la technique, puisqu'ils forcent à penser aux manières de bien diffuser des données, une question qui sont éminemment technique. La conception de cette API, et la réflexion qu'elle engendre sur les manières de diffuser des données de recherche, montrent que le partage de données n'est pas uniquement un problème technique, qui dépend de la disponibilité d'outils. Ce n'est pas non plus seulement un problème légal, où des licences encadrant clairement le partage et l'utilisation de données par autrui sont nécessaires. La diffusion de données de recherche est avant tout un problème scientifique, qui ne peut être résolu que par une bonne compréhension de ce qui est partagé. Il apparaît, au final, qu'il n'existe pas vraiment de données « brutes » : celles-ci demandent à être éditorialisées et incluses dans des fichiers structurés et documentés afin d'être réutilisables. Même en essayer de s'éloigner du texte (les catalogues encodés) pour arriver à diffuser des données, le processus de modélisation et rappelle l'importance de la structuration des données, qu'il s'agisse de texte ou de statistiques. Il est impossible de « seulement » diffuser des données, et on en revient donc toujours à ce que Flanders et Jannidis disent dans leur introduction au livre *The Shape of Data in the Digital Humanities* (2019) :

> Les processus de modélisation inscrivent en des termes formels notre connaissance du contenu et de la sémantique de nos données à l'intérieur de ces mêmes données. [4]

4. Fotis Jannidis et Julia Flanders, « Data modelling in a digital humanities context », dans *The shape of data in the digital humanities : modeling texts and text-based resources*, dir. Julia Flanders et Fotis Jannidis, London ; New York, 2019 (Digital research in the arts and humanities), p. 3-25, p. 9. Traduction de l'auteur. Original : « In effect, modelling processes write our knowledge about the content and semantics of our data into that data in formal terms [...] ».

Conclusion

Ainsi, en s'appuyant sur une connaissance du corpus, il est possible d'approximer une véritable compréhension « humaine » d'un texte semi structuré, en utilisant uniquement des expressions régulières. Si il n'est pas possible d'identifier le sens des différents éléments, il est possible de localiser et d'extraire les éléments signifiants. Il apparaît alors que ces éléments signifiants sont, d'une certaine manière, des formes, ou des motifs qui peuvent être détectés automatiquement ; la signification peut être identifiée en fonction du type de donnée extraite (un nombre est une date). Mais, dans un corpus semi-structuré, il surtout est possible d'inférer du « sens » d'un élément à partir de sa position dans le texte. C'est dans ce cadre qu'un encodage en `XML-TEI` des documents prend tout son intérêt : les différentes parties d'un texte sont balisées sémantiquement, ce qui est essentiel au traitement automatisé du texte : il devient possible de traiter le texte non pas dans son intégralité, mais au niveau d'une unité sémantique très précise, comme le nom donné à un manuscrit (le `tei:name`) ou une brève description (le `tei:trait`)

Le fait de travailler sur des corpus semi-structurés permet également de mettre en approche des méthodes alternatives, « low-tech », pour le traitement du langage et l'analyse textuelle dans des projets de recherche. Avec une bonne connaissance du corpus et des données structurées, il est possible de se baser entièrement sur des solutions techniquement « simples », telles que la détection de motifs à l'aide d'expressions régulières. Plutôt que de s'appuyer sur de l'apprentissage machine, et donc des solutions complexes, il est intéressant de s'appuyer sur des méthodes alternatives et de développer des algorithme plus complexes, basés sur une approche modulaire, qui sont capables de cibler très précisément les éléments d'un texte pertinent pour des problématiques de recherche précises. Ces méthodes de basse technologie, qui s'intéressent à un usage intelligent des technologies plutôt qu'à des technologies « intelligentes » forment un contre modèle qui gagnerait à être mis en avant dans les projets de recherche. L'intelligence artificielle devenue un outil central et pertinent dans les humanités numériques – à la base de la reconnaissance optique de caractères. Mais elle ne doit pas seulement être une nouveauté que les projets de recherche doivent à tout prix incorporer. C'est également une technologie extrêmement polluante [5],

5. Strubell, Ganesh et McCallum (E. Strubell, A. Ganesh et A. McCallum, « Energy and Policy Considerations for Deep Learning in NLP »…) ont étudié l'impact écologique des modèles d'apprentissage profond en traitement du langage naturel. En étudiant les ressources nécessaires à l'entraînement d'un modèle, les auteur.ice.s montrent (*Ibid.*, p. 1) que la mise au point d'un modèle *Transformer* peut émettre

utilisée de façon massive par des grands groupes industriels[6] et un argument de vente pour de nombreuses start-up, alors que les applications pratiques de l'intelligence artificielle soulèvent de graves problèmes éthiques[7]. La recherche en humanités numériques, fondée sur une connaissance des techniques et des documents utilisés, devrait fournir un espace pour remettre en question de telles technologies et pour développer des solutions alternatives, où la connaissance scientifique supplante l'intelligence machine. De façon modeste, le projet *MSS / Katabase* offre une telle remise en question : en s'appuyant sur une bonne connaissance de corpus textuels semi-structurés, le projet a développé une chaîne de traitement entièrement basée sur la détection de motifs. S'appuyer sur des méthodes de basse technologie ne vaut bien sûr pas seulement comme une déclaration de bonne foi : plus une solution est techniquement simple et accessible, plus elle est réutilisable, adaptable à d'autres jeux de données et plus facilement elle peut évoluer pour incorporer de nouvelles fonctionnalités.

pollution des large language models : https ://arxiv.org/abs/1906.02243

— parler du minimal computing ??

jusqu'à 626155 livres de CO_2, soit cinq fois la consommation d'une voiture durant toute sa vie (126000 livres de CO_2) et 17 fois les émissions de CO_2 émises par une personne américaine en un an (36156 livres). Il.elle.s rappellent également que les modèles d'apprentissage machine les plus performants sont aussi les plus gourmands en ressources et qu'un modèle doit être entraîné parfois plusieurs milliers de fois pour atteindre des scores satisfaisants (*Ibid.*). Avec l'augmentation générale des performances des modèles de traitement du langage naturel, l'augmentation marginale de leur performance est de plus en plus coûteuse en ressources : l'augmentation du score BLEU d'un modèle de traduction automatique de 0.1 points de performance (pour atteindre un score de 29,7) a coûté 150000 dollars en ressources environnementales et en coût de calcul (*Ibid.*, p. 4).

6. *Google*, et sa filiale *Youtube*, s'appuient depuis le milieu des années 2010 de plus en plus massivement sur l'intelligence artificielle (P. Covington, J. Adams et E. Sargin, « Deep Neural Networks for YouTube Recommendations »…, p. 1). L'algorithme de suggestion de vidéos sur *Youtube* repose entièrement sur de l'apprentissage machine(*ibid.*) : en fonction de l'historique de l'utilisateur.ice, l'algorithme « apprend » à suggérer du contenu pertinent. L'usage massif de telles technologies est particulièrement polluant : pour pouvoir suggérer des vidéos pertinentes, il faut, à chaque connexion de l'utilisateur.ice, consulter deux très grandes bases de données : dans un premier temps, l'algorithme sélectionne toutes les vidéos potentiellement pertinentes, depuis l'ensemble des vidéos disponibles sur le site ; ensuite, ces vidéos sont classées en consultant une seconde base de données et en filtrant les résultats obtenus par le comportement récent de l'utilisateur.ice (ce qu'il ou elle a recherché, combien de temps il ou elle a passé à regarder une vidéo, sa réaction à celle-ci...). Le 2 décembre 2020, *Google* a licencié la docteure Timnit Gebru, alors à la tête de l'équipe spécialisée en éthique de l'intelligence artificielle, suite à la rédaction d'un article sur les impacts écologiques de l'utilisation massive par l'entreprise de modèles de traitement automatisé du langage. Ce phénomène alarmant rappelle au passage la non-neutralité des publications scientifiques provenant ce cette entreprise, dont deux ont été citées dans le présent mémoire (T. Mikolov, K. Chen, G. Corrado, *et al.*, « Efficient Estimation of Word Representations in Vector Space »… ; P. Covington, J. Adams et E. Sargin, « Deep Neural Networks for YouTube Recommendations »…).

7. À ce sujet, voir par exemple le projet *Awful AI* de David Dao, qui liste une quantité impressionnante d'utilisations préoccupantes de telles technologies.

Annexes

Lien vers les dépôts en ligne de la chaîne de traitement *Katabase*

L'intégralité du code produit pour le projet *Katabase*, depuis la création des documents `TEI` jusqu'à l'application Web, sont disponibles sur la plateforme d'hébergement de code GitHub.

— Plateforme du projet

— Première étape : normalisation des documents `TEI`

— Deuxième étape : extraction de données et normalisation

— Troisième étape : enrichissement de données à l'aide de *Wikidata*

— Quatrième étape : création de jeux de données `JSON` pour le site web

— Application web *Katabase*

L'intégralité du code source est disponible sous licence libre (GNU GPL v3.0, MIT ou Creative Commons). Le site web peut également être consulté en ligne à cette adresse.

Documentation des différentes étapes

Ce chapitre contient la documentation des différentes étapes de la chaîne de traitement (en anglais), telle qu'elle se trouve sur les dépôts en ligne du projet sur GitHub. Cette documentation a simplement été convertie depuis le format `Markdown` vers LaTeX.

Output Data - level 1

This repository contains digitised manuscripts sale catalogs encoded in XML-TEI at level 1.

The data have not been cleaned (level 2) or post-processed (level 3).

Description of the data

Basic bibliographic information for each catalogue are available here.

Schema

You can find the ODD that validates the encoding in the repository Data_extraction (folder `_schemas`).

Workflow

Creation of the data

The creation process is described in detail in the following repo.

Cleaning the data

Entries of catalogues look like the following :

```
<item n="80" xml:id="CAT_000146_e80">
   <num>80</num>
   <name type="author">Cherubini (L.),</name>
   <trait>
   <p>l'illustre compositeur</p>
   </trait>
   <desc>L. a s.; 1836, 1 p 1 /2 in8.</desc>
   <measure commodity="currency" unit="FRF" quantity="12">12</measure>
</item>
```

Most of the reconciliation process uses data from the `<desc>` element of our xml files. We therefore need to correct typos to ease further post-processing, _e.g._ * `L. a s.` -> `L. a. s.` * `in8` -> `in-8` * `1 /2` -> `1/2` * `1 p` -> `1 p.`

The `clean_xml.py` script available here tackles this problem.

Installation and use

```
* git clone https://github.com/katabase/1_OutputData.git
* cd 1_OutputData
* python3 -m venv my_env
* source my_env/bin/activate
* pip install -r requirements.txt
* python script/clean_xml.py -f FILENAME processes one single file
    OR
* python script/clean_xml.py -d DIRECTORY processes all the files
 ↪  contained in a directory
```

Credits

* The ODD was created by Lucie Rondeau du Noyer.

* `clean_xml.py`was created by Simon Gabay.

* The catalogs were encoded by Lucie Rondeau du Noyer, Simon Gabay, Matthias Gille Levenson, Ljudmila Petkovic and Alexandre Bartz.

Cite this repository

Alexandre Bartz, Simon Gabay, Matthias Gille Levenson, Ljudmila Petkovic and Lucie Rondeau du Noyer, _Manuscript sale catalogues_, Neuchâtel : Université de Neuchâtel, 2019, https ://github.com/katabase/1_OutputData.

Licence

Cleaned Data - level 2

This repository contains digitised manuscripts sale catalogs encoded in XML-TEI at level 2.

The data have been cleaned (level 2) but not post-processed (level 3) yet.

Schema

You can find the ODD that validates the encoding in the repository Data_extraction (folder `_schemas`).

Workflow

Once the data have been cleaned, we can start to extract information from the `desc`.

`extractor-xml.py` extracts informations and then retrieves them in the same XML file (level 3).

The script transforms this

```
<item n="80" xml:id="CAT_000146_e80">
   <num>80</num>
   <name type="author">Cherubini (L.),</name>
   <trait>
   <p>l'illustre compositeur</p>
   </trait>
   <desc>L. a. s.; 1836, 1 p. in-8.</desc>
   <measure commodity="currency" unit="FRF" quantity="12">12</measure>
</item>
```

into

```xml
<item n="80" xml:id="CAT_000146_e80">
   <num>80</num>
   <name type="author">Cherubini (L.),</name>
   <trait>
   <p>l'illustre compositeur</p>
   </trait>
   <desc>
   <term>L. a. s.</term>;<date>1836</date>,
       <measure type="length" unit="p" n="1">1 p.</measure>
       <measure unit="f" type="format" n="8">in-8</measure>.
       <measure commodity="currency" unit="FRF"
        ↪  quantity="12">12</measure>
   </desc>
</item>
```

To carry this task we use `extractor_xml.py` [available here].

Installation and use

```
* git clone https://github.com/katabase/2_CleanedData.git
* cd 2_CleanedData
* python3 -m venv my_env
* source my_env/bin/activate
* pip install -r requirements.txt
* cd script
* python3 extractor_xml.py directory_to_process
```

Note that you have to be in the folder `script`to execute `extractor_xml.py` and that the script only works with filenames ending with `_clean.xml` (files must have been beforehand cleaned).

The output files will be in the folder `output`.

Credits

* Scripts were created by Matthias Gille Levenson and improved by Alexandre Bartz with the help of Simon Gabay. * The catalogs were encoded by Lucie Rondeau du Noyer, Simon Gabay, Matthias Gille Levenson, Ljudmila Petkovic and Alexandre Bartz.

Cite this repository

Alexandre Bartz, Simon Gabay, Matthias Gille Levenson, Ljudmila Petkovic and Lucie Rondeau du Noyer, _Manuscript sale catalogues_, Neuchâtel : Université de Neuchâtel, 2020, https ://github.com/katabase/2_CleanedData.

Licence

LEVEL 3 - WIKIDATA ENRICHMENTS

Presentation

This part of the pipeline

— reconciles the names in our dataset with wikidata IDs

— runs the same 4 `sparql` requests on all IDs

— stores the output of the `sparql` requests in a `sparql` file

— updates the `tei` files with the wikidata IDs

The aim is to produce normlised data to connect to catalogue entries, in order to understand our dataset better and to isolate the factors detrmining a price.

Installation, pipeline and use

In short

With the proper python virtual environment sourced, without running tests, just type :

```
shell
python main.py -n # build the input data table
python main.py -i # align tei:names with wikidata entities
python main.py -s # run sparql queries on those entities
python main.py -w # reinject the wikidata ids into the tei catalogues
```

Even simpler, you can just run the below script :

```
shell
bash pipeline.sh
```

Installation

This works on MacOS and Linux (ubuntu, debian based distributions).

```
shell
git clone https://github.com/katabase/3_WikidataEnrichment # clone the repo
cd 3_WikidataEnrichment # move to the dictory
python3 -m venv env # create a python virtualenv
source env/bin/activate # source python from the virtualenv
pip install -r requirements.txt # install the necessary librairies
```

Pipeline and use

All scripts run by running `main.py` with a specific argument. 4 GBs of RAM are recommended to run the scripts.

As a reminder, here is catalogue entries'`tei` structure :

```
<item n="80" xml:id="CAT_000146_e80">
   <num>80</num>
   <name type="author">Cherubini (L.),</name>
   <trait>
   <p>l'illustre compositeur</p>
   </trait>
   <desc>
   <term>L. a. s.</term>;<date>1836</date>,
   <measure type="length" unit="p" n="1">1 p.</measure>
   <measure unit="f" type="format" n="8">in-8</measure>.
   <measure commodity="currency" unit="FRF" quantity="12">12</measure>
   </desc>
</item>
```

Step 1 : create an input TSV - `python main.py -n`

The first step is to create a `tsv` file that will be used to retrieve the wikidata IDs :

— the `tsv` is made of 5 columns (see example below) :

  — `xml id` : the item's `xml:id`

  — `wikidata id` : the wikidata ID (to be retrieved in the next step)

  — `name` : the `tei:name` of that item

  — `trait` : the `tei:trait` of that item

xml id	wikidata id	name	trait
CAT_000362_e27086		ADAM (Ad.)	célèbre compositeur de musique.

— **running this step** :

```
shell
python main.py -n
```

Step 2 : retrieve the wikidata IDs - `python main.py -i`

The wikidata IDs are retrieved by running a full text search using the wikidata API.

— the **algorithm functions** as follows :

- — the input is file created at the previous step (`script/tables/nametable_in.tsv`). The `name` and `trait` columns are used to create data for the API search
- — two columns are processed to prepare the data for the API search :
  - — from the `name`, we determine the kind of `name` we're working with (the name of a person, of a nobility, of an event, of a place...). This determines different behaviours.
  - — the `name` is normalized : we extract and translate nobility titles, locations... First and last names are extracted. If the first name is abbreviated, we try to rebuild a full name from its abbreviated version.
  - — the `trait` is processed to extract and translate occupations, dates...
  - — the output is stored in a dictionnary
- — this `dict` is passed to a second algorithm to run text searches on the API. Depending on the data stored in the dict, different queries are ran. A series of queries are run until a result is obtained
- — finally, the result is written to a TSV file (`out/wikidata/nametable_out.tsv`). Its structure is the same as that of `nametable_in`, with some changes. Here are the column names :
  - — `tei:xml_id` : the `@xml:id` from the `tei` files
  - — `wd:id` : the wikidata ID
  - — `tei:name` : the `tei:name`
  - — `wd:name` : the name corresponding to the wikidata ID (to ease the data verification process)
  - — `wd:snippet` : a short summary of the wikidata page (to ease the data verification process)
  - — `tei:trait` : the `tei:trait`
  - — `wd:certitude` : an evaluation of the degree of certitude (whether we're certain that the proper id has been retrieved)
- — once this script has completed, a deduplicated list of wikidata IDs is written to `script/tables/id_wikidata.txt`. This file will be used as input for the next step.

- the F1 score for this step (evaluating the number of good wikidata IDs retrieved) is `0.674`, based on tests run on 200 items.
- this step takes a lot of time to complete, but, thanks to log files, the script can be interrupted and restarted at any point.

— **running this step** :

```
shell
python main.py -i
```

Step 3 : running `sparql` queries - `python main.py -s`

— the **algorithm** is much simpler : for each wikidata ID, 4 sparql queries are run. The results are returned in `json` or, if there's a mistake, `xml`. The results are translated to a simpler `json` and the result is stored to `out/wikidata/wikidata_enrichments.json`. This step takes a lot of time, but the script can be stopped and continued at any point.

— the **output structure** is as follows (each key is mapped to a list of results ; the list can be empty ; the empty lines in the dict separates the different wikidata queries) :

```
out = {'instance': [], 'instanceL': [], # what "category" an id belongs to (person, litterary work...)
'gender': [], 'genderL': [], # the gender of a person
'citizenship': [], 'citizenshipL': [], # citizenship
'lang': [], 'langL': [], # languages spoken
'deathmanner': [], 'deathmannerL': [], # the way a person died
'birthplace': [], 'birthplaceL': [], # the place a person is born
'deathplace': [], 'deathplaceL': [], # the place a person died
'residplace': [], 'residplaceL': [], # the place a person lived
'burialplace': [], 'burialplaceL': [], # where a person is buried

'educ': [], 'educL': [], # where a person studied
'religion': [], 'religionL': [], # a person's religion
'occupation': [], 'occupationL': [], # general description of a person's occupation
'award': [], 'awardL': [], # awards gained
'position': [], 'positionL': [], # precise positions held by a person
'member': [], 'memberL': [], # institution a person is member of
'nobility': [], 'nobilityL': [], # nobility titles
'workcount': [], # number of works (books...) documented on wikidata
'conflictcount': [], # number of conflicts (wars...) a person has participated in
'image': [], # url to the portrait of a person
'signature': [], # url to the signature of a person
'birth': [], 'death': [], # birth and death dates
'title': [], # title of a work of art / book...
'inception': [], # date a work was created or published
'author': [], 'authorL': [], # author of a book
'pub': [], 'pubL': [], # publisher of a work
'pubplace': [], 'pubplaceL': [], # place a work was published
'pubdate': [], # date a work was published
'creator': [], 'creatorL': [], # creator of a work of art
'material': [], 'materialL': [], # material in which a work of art is made
'height': [], # height of a work of art
'genre': [], 'genreL': [], # genre of a work or genre of works created by a person
'movement': [], 'movementL': [], # movement in which a person or an artwork are inscribed
'creaplace': [], 'creaplaceL': [], # place where a work was created
'viafID': [], # viaf identifier
'bnfID': [], # bibliothèque nationale de france ID
'isniID': [], # isni id
'congressID': [], # library of congress identifier
'idrefID': [] # idref identifier}
```

— **running this step** :

```
python main.py -s
```

Step 4 : reinject the wikidata ids into the TEI catalogues - `python main.py -w`

— all `tei:items` are linked with a wikidata ID retrieved during the process.

— the wikidata IDs are included in a `@key` attribute inside the `tei:name` and prefixed by the token `wd:`.

— a pattern to handle this prefix is provided in the `tei:teiHeader`, in the `tei:editorialDecl//tei` this allows to automatically rebuilt a URL to the proper wikidata page.

— the output is written to `out/catalogues`.

Running tests - `python main.py -t`

— the tests are only run on the **step 2** (for the rest, we are certain of the result).

— They are based on 200 catalogue entries. The test dataset ressembles the full dataset (about as many different kinds of entries, from different catalogues, with as many `tei:trait`s as in the main dataset)

— Several tests are run. Two tests are testing isolate parameters of the dictionnary built in the step 1 and the efficiency of the function that rebuilds the first name from its abbreviation. The other tests are for the final algorithm and they build statistics it. They also calculate its execution time using different parameters.

— **running the tests** :

```
python main.py -t
```

Other options :

— **counting the most used words in the `tei:traits`** of the input dataset (to tweak the way the dictionnary is built in the step 2) : `python main.py -c`

— **`python main.py -x`** : a throwaway option to map to a function in order to use a script that is not accessible from the above arguments

---

Summarizing, the options are

```
* -c --traitcounter : count most used terms in the tei:trait (to tweak the
    matching tables)
* -t --test : run tests (takes ~20 minutes)
* -i --wikidataids : retrieve wikidata ids (takes up to 10 to 20 hours!)
```

```
* -s --runsparql : run sparql queries (takes +-5 hours)
* -n --buildnametable: build the input table for -i --wikidataids (a table
    from which
   to retrieve wikidata ids
* -x --throwaway : run the current throwaway script (to test a function or
    whatnot)
```

Credits

Scripts developped by Paul Kervegan in spring-summer 2022.

License

Tagged Data - level 3

This repository contains digitised manuscripts sale catalogs encoded in XML-TEI at level 3.

The data have been cleaned (level 2) and post-processed (level 3).

Schema

You can find the ODD that validates the encoding in the repository Data_extraction (folder `_schemas`).

Workflow

Once the data have been cleaned and post-processed, we can check them. Some errors may appear and some corrections may be needed.

From this data, `extractor-json.py` extracts informations and retrieves them in an JSON file, available here.

The script transforms this

```
<item n="80" xml:id="CAT_000146_e80">
   <num>80</num>
   <name type="author">Cherubini (L.),</name>
   <trait>
   <p>l'illustre compositeur</p>
   </trait>
   <desc>
   <term>L. a. s.</term>;<date>1836</date>,
   <measure type="length" unit="p" n="1">1 p.</measure>
   <measure unit="f" type="format" n="8">in-8</measure>.
   <measure commodity="currency" unit="FRF" quantity="12">12</measure>
   </desc>
</item>
```

into

From `export.json`, we can proceed at the reconciliation of the catalogues entries.

If you want to learn more about the reconciliation, visite this repository.

If you want to query the database, don't hesitate to try our application.

Installation and use

Note that you have to be in the folder `scriptto` execute `extractor_json.py`.

```
{"CAT_000146_e80_d1": {"desc": "L. a. s.; 1836, 1 p. in-8. 12",
   "price": 12.0,
   "author": "Cherubini",
   "date": "1836",
   "number_of_pages": 1.0,
   "format": 8,
   "term": 7,
   "sell_date": "1893-03"}}
```


```
* git clone https://github.com/katabase/3_TaggedData.git
* cd 3_TaggedData
* python3 -m venv my_env
* source my_env/bin/activate
* pip install -r requirements.txt
* cd script
* python3 extractor_json.py
```

The output file, `export.json`, is in the folder `output`.

Unittest

If you want run some unittests, try in the folder `script` :

```
python3 test.py
```

Credits

* The script was created by Alexandre Bartz with the help of Matthias Gille Levenson and Simon Gabay. * The catalogs were encoded by Lucie Rondeau du Noyer, Simon Gabay, Matthias Gille Levenson, Ljudmila Petkovic and Alexandre Bartz.

Cite this repository

Alexandre Bartz, Simon Gabay, Matthias Gille Levenson, Ljudmila Petkovic and Lucie Rondeau du Noyer, _Manuscript sale catalogues_, Neuchâtel : Université de Neuchâtel, 2020, https ://github.com/katabase/3_TaggedData.

Licence

Application

This repository contains the web publication application of the corpus of manuscript sales catalogues. This branch contains the current stable version of the website. See `versionX.X.X` for the older versions.

Getting started :

— First, download this repository. Using command lines, clone the repository with :

```
git clone https://github.com/katabase/Application.git
cd Application
```

— Then, create a virtual environment and activate it :

```
python3 -m venv my_env
source my_env/bin/activate
```

— Now, you have to install dependencies :

```
pip install -r requirements.txt
```

— You can finally launch the application :

```
python3 run.py
```

Use the `KatAPI`

`KatAPI` is an API that allows the automated retrieval of data from the project in `json` or `xml-tei`. The API allows to revtrieve catalogue entries by author, sale date and original date of the manuscript ; it also allows to retrieve a complete catalogue, or statistics on one or several catalogues. Finally, it creates custom error messages in `json` of `tei` if an error occurs. For a more complete description, please see the Katabase website.

Quick start

The endpoint for the API is **`https://katabase.huma-num.fr/katapi?`**. The arguments provided by the client are added after this endpoint ; the application will process thoses arguments and send back a response in the requested format (`json` or `xml-tei`, the default being `json`). If there is an error on the client side (unauthorized parameters or values) or on the server side (unexpected error), a response will be issued in `json` or `xml-tei` (depending on the client's request) describing the query parameters, the time of the query and the error that occured.

Possible query parameters and authorized values

HTTP methods

The only **authorized HTTP method** is `GET`.

Possible parameters

The **possible parameters** are :

- **`format`** : the format of the API's response body. Possible values are :
  - `json` : **this is the default value**.
  - `tei` : return an `xml-tei` response.
- **`level`** : the requested data's level. Possible values are :
  - `item` : data is retrieved at item level. **This is the default value.**
  - `cat_data` : statistical data on one or several catalogues will be retrieved
    - this value is incompatible with the `orig_date` parameter.
  - `cat_full` : a complete catalogue encoded in `xml-tei` will be retrieved
    - if this value is provided, then the only other authorized parameters are `format=tei` and `id` (with `id` matching `CAT-\d+`).
- **`id`** : the identifier of the item or catalogue(s) to retrieve (depending on the value of `level`). If this parameter is provided, data will only be retrieved for a single catalogue or catalogue entry. This parameter cannot be used together with the `name` parameter. Possible values are :
  - if the query is at item level, a catalogue entry's `@xml:id`. This identifier is a string that matches the pattern : `CAT_\d+_e\d+_d\d+`.
  - the query is run at catalogue level (`level=cat_full` or `level=cat_data`), a catalogue's `@xml:id`. This identifier is a string that matches the pattern : `CAT_\d+`.
- **`name`** : if the `id` parameter is not supplied, the name of the catalogue(s) or catalogue entry(ies) to retrieve. Note that this parameter can, and will, return several items. Possible values :

— if `level=item`, the `tei:name` being queried. Only the last name in the `tei:name` is indexed in the search engine and only this one will yield a result. If a first name and a last name are provided, no result can be yield, since the first name is not indexed.

— if `level=cat_stat`, the catalogue type (to be found in `(TEI//sourceDesc/bibl/@ana` in the `xml` representation of a catalogue). Possible values are :

— "LAC" : Vente Jacques Charavay,

— "RDA" : Revue des Autographes,

— "LAV" : Catalogue Laveredet,

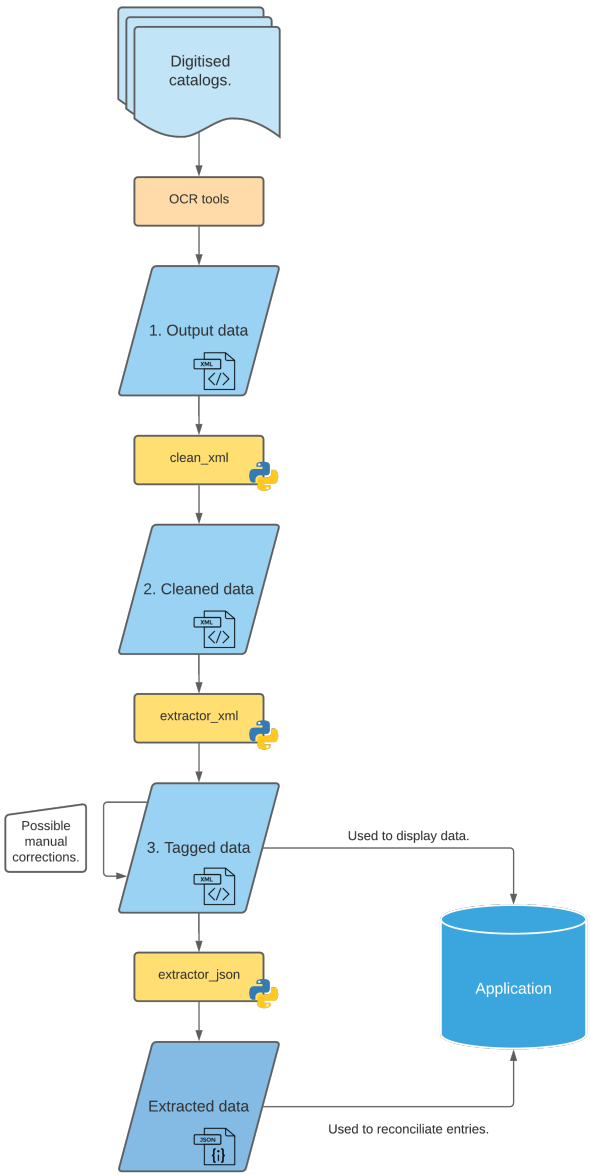
— "AUC" : Auction sale

— "OTH" : not yet in use in our dataset

— **`sell_date`** : the sale date for a manuscript or a catalogue. Values must match the regular expression `\d{4}(-\d{4})?` : a year in `YYYY` format or a year range in `YYYY-YYYY` foramat.

— **`orig_date`** : the date a manuscript item was created. This parameter is only authorized if `level=item`. Values must match the regular expression `\d{4}(-\d{4})?` : a year in `YYYY` format or a year range in `YYYY-YYYY` foramat.

Workflow



Website updates and description of the git branches

The structure of the git repository is as follows :

— `main` for the current, stable version of the Katabase app

— `dev` for the unstable version of the app, in developpment and not running online.

— `versionX.X.X` are archive repositories to document the former versions of the Katabase app. There should be as many of these branches as there are new versions of the website, and their `X.X.X` code should follow the release numbers.

New additions to the website should be done on `dev` and tested before being moved to `main`. The version of the website visible on `main` should be the same as the version of the website online (unless, for reasons out of our control, we can't publish a new version of the website online, but a new version is ready and won't be changed again). Before merging a new version of the website from `dev` to `main`, the `main` branch should be moved the `versionX.X.X`. A new release should then be created for the updated version of the website.

Credits

The application was designed by Alexandre Bartz and Paul Kervegan with the help of Simon Gabay, Matthias Gille Levenson and Ljudmila Petkovic.

Cite this repository

Licence

Résultat des tests de l'algorithme d'extraction d'informations de *Wikidata*

Les résultats des tests présentés dans les tables ci-dessous (1, 2) ont été menés sur un jeu de 200 couples `tei:name` et `tei:trait`. Ils ont été choisis de façon à être représentatifs du jeu de données complet, avec une variété d'entrées semblables (personnes nobles et non nobles, entrées qui ne sont pas consacrées à des personnes...). Ce jeu de test présente également une structure et un degré de détail semblable au jeu de données complet. Il y a notamment exactement la même proportion d'entrées présentant un `tei:trait` que dans le jeu de test et le jeu final.

Les résultats en pourcentage sont toujours exprimés en **pourcentages du nombre total d'entrées requêtées** (soit 200).

Type de requête	**Noms**	**Noms et nom de famille noble**	**Noms et titre de noblesse**	**Noms et date de naissance et de mort**	**Noms et occupation**
Avec reconstitution des prénoms	48%	45.9%	50%	52.6%	53.1%
Sans reconstitution des prénoms	42%	40.5%	50%	52.6%	53.1%

TABLE 1 – Tests menés avec un jeu de 200 entrées sur des paramètres isolés

Le tableau ci-dessus (1) présente les résultats de tests menés pour isoler l'impact de chaque paramètee dans l'obtention du bon résultat. Les mêmes requêtes ont été lancées sur un jeu de test de 200 entrées de catalogue. Toutes les requêtes contient le prénom et le nom de famille complets (« Noms » dans la table). La première requête ne contient que ce paramètre, tandis que toutes les autres requêtes sont lancées avec un paramètre en plus : titre de noblesse, nom de famille noble, dates et occupation. Le fait que ces paramètres soient utilisés ne garantit pas qu'ils soient disponibles dans le jeu de données utilisé en entrée. Ce test permet d'isoler de mieux construire l'algorithme final de recherches en plein texte sur le moteur de recherche de *Wikidata*, puisqu'il permet de voir quels sont les paramètres les plus fiables.

Bons identifiants récupérés	65%
Score F1	0.674
Précision	0.677
Rappel	0.670
Résultats certains	36%
Faux positifs parmi les éléments certains	8.5%
Total d'entrées requêtées	200
Identifiants récupérés	192
Silence	8
Temps d'exécution avec l'option `fetch`	88.3 secondes
Temps d'exécution sans l'option `fetch`	92.49 secondes

TABLE 2 – Tests menés avec un jeu de 200 entrées sur l'algorithme final

Le tableau ci-dessus présente les résultats d'un test évaluant les performances de l'algorithme final. Voici une explication de ces résultats :

— *Bons identifiants récupérés* : la proportion, exprimée en pourcentage, du nombre d'identifiants récupérés par l'algorithme qui correspondent aux identifiants récupérés manuellement

— *Score F1* : le score F1, soit la moyenne pondérée de la précision et du rappel. Contrairement à la mesure ci-dessus, le score F1 prend en compte le silence et les faux négatifs, et offre donc une interprétation plus complète de la qualité du script.

— *Précision* : la précision est une composante du score F1 qui mesure la proportion d'identifiants pertinents obtenus parmi tous les identifiants obtenus.

— *Rappel* : le rappel est une composante du score F1 qui mesure la proportion d'identifiants pertinents obtenus parmi l'ensemble des identifiants pertinents qui existent.

— *Résultats certains* : pour accélérer le processus de relecture, un score de certitude a été attribué à certains identifiants obtenus grâce à l'algorithme. Ce score dépend du nombre de paramètres utilisés dans la recherche qui a permis de récupérer l'identifiant. 36% des résultats obtenus sont considérés comme certains.

— *Faux positifs parmi les éléments certains* : le score de certitude présenté ci-dessus admet un taux d'erreur qui est ici quantifié : 8.5% des résultats obtenus sont erronnés alors qu'ils ont été considérés comme certains (8,5% du jeu de données total, donc).

— *Total d'entrées requêtées* : le volume du jeu de données de test, soit 200 entrées.

— *Identifiants récupérés* : le nombre d'identifiants obtenus, qu'ils soient corrects ou non.

— *Silence* : le nombre d'entrées pour lesquelles aucun identifiant n'a été obtenu.

— *Temps d'exécution avec l'option `fetch`* : le temps pris, en secondes, pour lancer l'intégralité de l'algorithme de récupération d'identifiants *Wikidata* sur les 200 entrées en stockant dans des fichiers de log les requêtes lancées et les identifiants requêtés.

— *Temps d'exécution sans l'option* `fetch` : le temps pris, en secondes, pour lancer l'intégralité de l'algorithme de récupération d'identifiants sur les 200 entrées sans utiliser de fichiers de log.

Base de connaissances / Score	Score F1 – recherche des candidates	Score F1 – Lien avec le candidat	Overall linking accuracy
DBPedia	0.899	0.536	0.834
DataBNF	0.689	0.726	0.7
Wikidata	0.869	0.611	0.85

TABLE 3 – Résultats du NEL de lieux dans des textes littéraires du XIXe s. par Soudani et al. (2018)

Scores obtenus par Aicha Soudani et al. pour le liage d'entités nommées pour des lieux dans des textes littéraires français du XIX$^{\text{ème}}$ s., présentés à la conférence *HumaNS'2018*[8]. Les scores F1 ne figurent pas dans l'article original, qui ne présente que la précision et le rappel. Je les ai donc calculés moi même. La méthode de calcul de l'*overall linking accuracy* (« exactitude générale du liage ») n'est pas précisée dans l'article ; il est seulement indiqué que « Cette mesure essaie d'évaluer l'efficacité globale du système, et non par phase, et exprime la fiabilité de REDEN pour la tâche de résolution des entités nommées. »[9].

La méthode utilisée dans cet article est la suivante : les auteur.ice.s s'appuient fortement sur l'apprentissage machine, en identifiant d'abord des entités dans le texte à l'aide de `SEM` avant de lier les entités à différentes bases de connaissance en utilisant `REDEN`.

8. A. Soudani, Y. Meherzi, A. Bouhafs, *et al.*, « Adaptation et évaluation de systèmes de reconnaissance et de résolution des entités nommées pour le cas de textes littéraires français du 19ème siècle »…, p. 4.

9. *Ibid.*

Graphiques

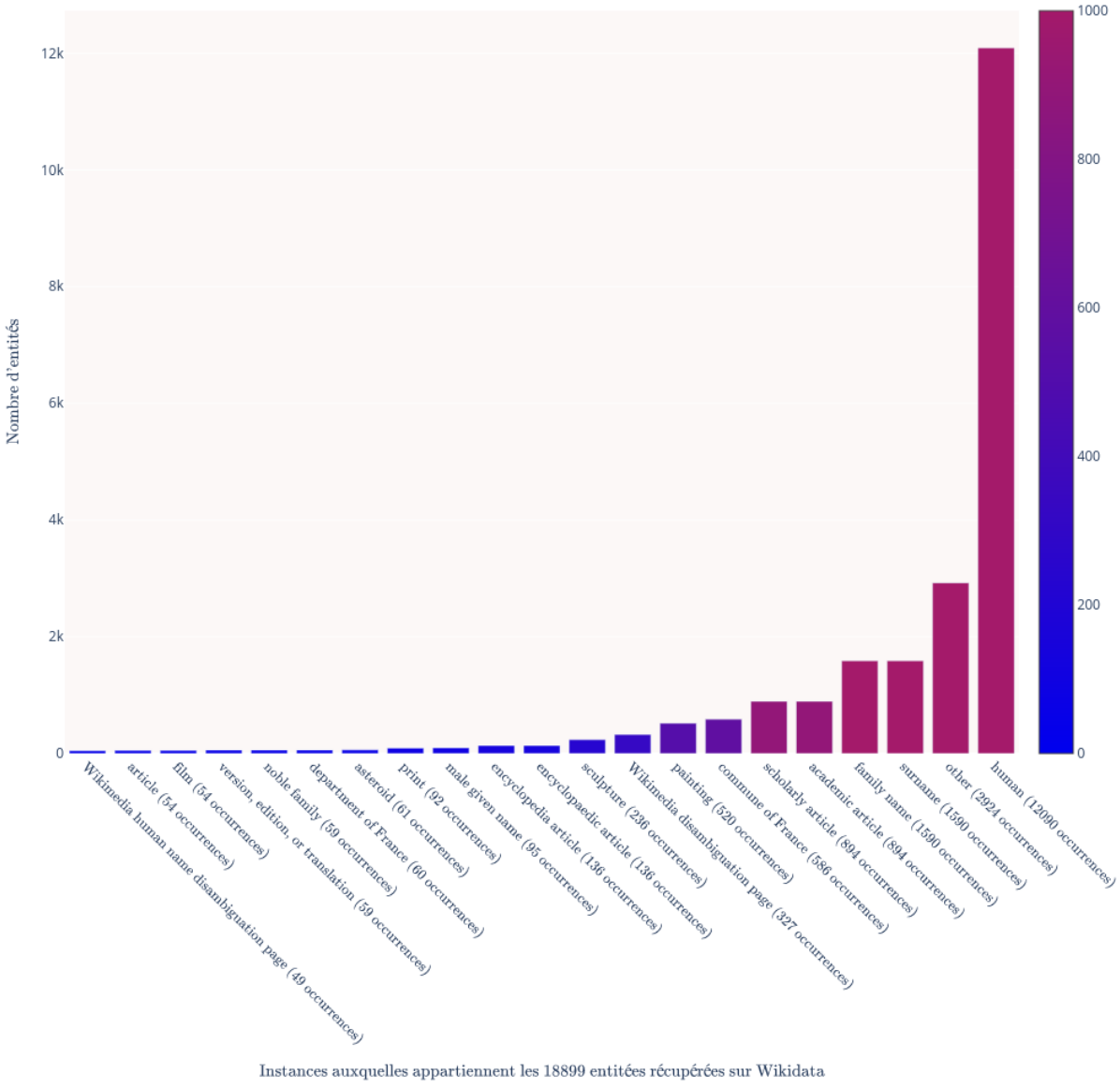


FIGURE 2 –
Occurrences des différentes catégories auxquelles appartiennent les entités *Wikidata* liées avec les entrées de catalogues

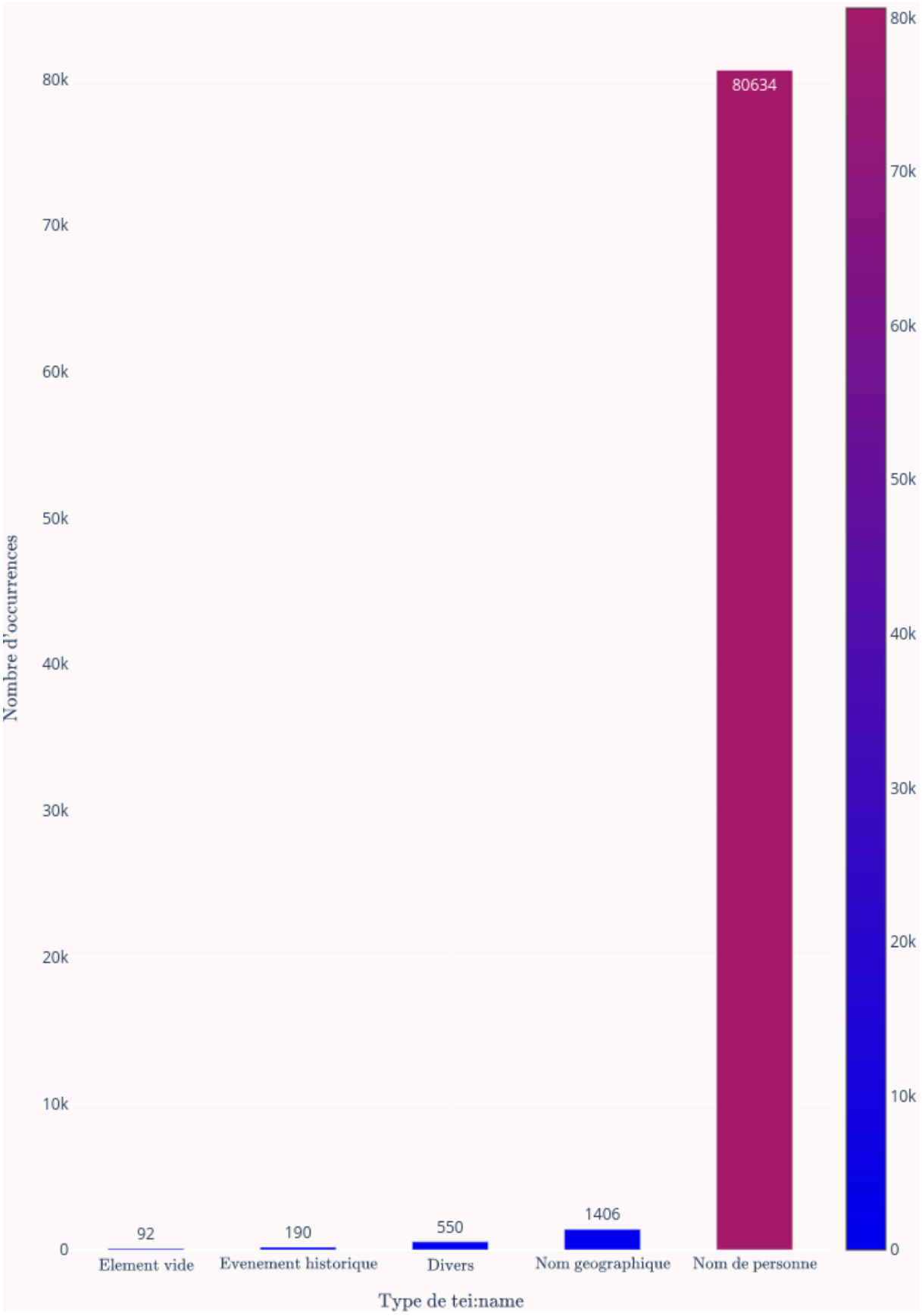


FIGURE 3 – Répartition des différents types de `tei:name`

Code source et données encodées

```
functions = {
    "général": "general",
    "maréchal": "marshal",
    "lieutenant": "military",
    "officier": "military",
    "colonel": "military",
    "lieutenant-colonel": "military",
    "commandant": "military",
    "capitaine": "military",  # "less important" military positions
    "roi": "king",
    "empereur": "emperor",
    "president": "president",
    "homme politique": "politician",
    "président de l'assemblée": "politician",
    "orateur": "politician",
    "député": "politician",
    "secrétaire d'état": "politician",
    "sénateur": "politician",
    "écrivain": "writer",
    "auteur": "writer",
    "romancier": "writer",
    "acteur": "actor",
    "actrice": "actress",
    "cantatrice": "singer",
    "chanteur": "singer",
    "chanteuse": "singer",
    "peintre": "painter",
    "sculpteur": "sculptor",
    "statutaire": "sculptor",
    "compositeur": "composer",
    "musicien": "musician",
    "musicienne": "musician",
    "tragédien": "actor",
    "chansonnier": "chansonnier",
    "achitecte": "architect",
    "journaliste": "journalist",
    "inventeur": "inventor",
    "chimiste": "chemist",
    "connétable": "constable",
    "archevêque": "archbishop",
    "évêque": "bishop",
    "docteur": "physicist",
    "médecin": "physicist"
}
```

Code source 1 – Table de conversion associant un métier à son équivalent normalisé

```
dpts = [
    "ain",
    "aisne",
    "allier",
    "basses-alpes",
    "hautes-alpes",
    "alpes-maritimes",
    "annepins",
    "provence",
    "ardèche",
    "ardennes",
    "arriège",
    "arno",
    "aube",
    "aude",
    "aveyron",
    "bouches-de-l'elbe",
    "bouches-de-l'escaut",
    "bouches-de-l'yssel",
    "bpuches-de-la-meuse",
    "bouches-du-rhin",
    "bouches-du-rhône",
    "bouches-du-weser",
    "calvados",
    "cantal",
    "charente",
    "charente-inférieure",
    "cher",
    "corrèze",
    "corse",
    "côte-d'or",
    "côtes-du-nord",
    "creuse",
    "deux-nèthes",
    "deux-sèvres",
    "doire",
    "dordogne",
```

```
    "doubs",
    "drôme",
    "dyle",
    "ems-occidental",
    "ems-oriental",
    "ems-supérieur",
    "escaut",
    "eure",
    "eure-et-loir",
    "finistère",
    "forêts",
    "gard",
    "haute-garonne",
    "gers",
    "gironde",
    "hérault",
    "ille-et-villaine",
    "indre",
    "indre-et-loire",
    "isère",
    "jemappes",
    "jura",
    "landes",
    "léman",
    "loire",
    "loir-et-cher",
    "haute-loire",
    "loire-inférieure",
    "loiret",
    "lot",
    "lot-et-garonne",
    "lozère",
    "lys",
    "maine-et-loire",
    "manche",
    "marengo",
    "marne",
    "haute-marne",
```

```
    "méditerrannée",
    "mayenne",
    "meurthe",
    "meuse",
    "meuse-inférieure",
    "mont-blanc",
    "mont-tonnerre",
    "montenotte",
    "morbihan",
    "meuse",
    "moselle",
    "nièvre",
    "nord",
    "oise",
    "ombrone",
    "orne",
    "ourte",
    "paris",
    "pas-de-calais",
    "pô",
    "puy-de-dôme",
    "hautes-pyrénées",
    "basses-pyrénées",
    "pyrénées-orientales",
    "haut-rhin",
    "bas-rhin",
    "rhin-et-moselle",
    "rhône",
    "rhône-et-loire",
    "roer",
    "rome",
    "haute-saône",
    "saône-et-loire",
    "sambre-et-meuse",
    "sarre",
    "sarthe",
    "seine",
    "seine-et-marne",
```

```
    "seine-et-oise",
    "seine-inférieure",
    "sézia",
    "simplon",
    "deux-sèvres",
    "somme",
    "stura",
    "tarn",
    "tarn-et-garonne",
    "taro",
    "trasimène",
    "var",
    "vaucluse",
    "vendée",
    "vienne",
    "haute-vienne",
    "vosges",
    "yonne",
    "yssel-supérieur",
    "zuyderzée"
]
```

Code source 2 – Liste de départements du XIXe s. pour détecter des informations géographiques

```
countries = {
    "états-unis d'amérique": "united states of america",
    "etats-unis d'amérique": "united states of america",
    "états unis d'amérique": "united states of america",
    "etats unis d'amerique": "united states of america",
    "états-unis": "united states of america",
    "etats-unis": "united states of america",
    "etats unis": "united states of america",
    "états unis": "united states of america",
    "italie": "italy",
    "grèce": "greece",
    "canada": "canada",
    "chine": "china",
    "haïti": "haiti",
    "tobago": "tobago",
    "brésil": "brasil",
    "burkina-faso": "burkina-faso",
    "cameroun": "cameroun",
    "tchad": "tchad",
    "congo": "congo",
    "gabon": "gabon",
    "guinée": "guinea",
    "côte d'ivoire": "ivory coast",
    "mali": "mali",
    "mauritanie": "mauritania",
    "niger": "niger",
    "sénégal": "senegal",
    "madagascar": "madagascar",
    "seychelles": "seychelles",
    "tanzanie": "tanzania",
    "zanzibar": "zanzibar",
    "liban": "lebanon",
    "syrie": "syria",
    "inde": "india",
    "laos": "laos",
    "viet-nâm": "vietnam"
}
```

Code source 3 – Table de conversion pour les pays

```
colonies = [
    "québec",
    "ontario",
    "saint-pierre-et-miquelon",
    "mississippi",
    "missouri",
    "louisiane",
    "anguilla",
    "antigua",
    "dominique",
    "saint-domingue",
    "guadeloupe",
    "monsterrat",
    "saint-martin",
    "saint-barthélémy",
    "sainte-lucy",
    "saint-vincent-et-les-grenadines",
    "saint-eustache",
    "saint-christophe",
    "martinique"
    "guyane française",
    "guyane",
    "maroc",  # unfortunately the morocco referred to in XIXth century
    ↪  france is a french protectorate
    "algérie",  # same
    "algérie française",  # same
    "tunisie",  # same
    "fezzan",
    "dahomey",
    "haute-volta",
    "oubangui-chari",
    "congo français",
    "moyen-congo",
    "guinée française",
    "soudan français",
    "gorée",
    "tigi",
```

```
    "djibouti",
    "cheikh saïd",
    "comores",
    "fort-dauphin",
    "îles maurice",
    "mayotte",
    "la réunion",
    "îles éparses",
    "île amsterdam",
    "île saint-paul",
    "archipel crozet",
    "îles kerguelen",
    "castellorizo",
    "grand-liban",
    "sandjak d'alexandrette",
    "indes françaises",
    "pondichéry",
    "karikal",
    "yanaon",
    "mahé",
    "chanderngor",
    "tonkin",
    "annam",
    "cochinchine",
    "guangzhou wan",
    "shanghai",
    "guangzhou",
    "tianjin",
    "hankou",
    "clipperton",
    "nouvelle-calédonie",
    "polynésie française",
    "vanuatu",
    "nouvelles-hébrides",
    "wallis et futuna"
]
```

Code source 4 – Liste d'anciennes colonies françaises utilisées pour la détection de motifs

```
provinces = [
    "armagnac",
    "île-de-france",
    "berry",
    "orléanais",
    "normandie",
    "languedoc",
    "lyonnais",
    "dauphiné",
    "champagne",
    "aunis",
    "saintonge",
    "poitou",
    "guyenne et gascogne",
    "bourgogne",
    "picardie",
    "anjou",
    "provence",
    "angoumois",
    "bourbonnais",
    "marche",
    "bretagne",
    "maine",
    "touraine",
    "limousin",
    "comté de foix",
    "auvergne",
    "béarn",
    "alsace",
    "artois",
    "roussillon",
    "flandre française et hainaut français",
    "franche-comté",
    "lorraine et trois-évêchés",
    "corse",
    "nivernais",
]
```

Code source 5 – Liste d'anciennes provinces françaises pour la détection de motifs

```
events = {
    "défense nationale": "government of national defense",
    "defense nationale": "government of national defense",
    "révolution française": "french revolution",
    "revolution francaise": "french revolution",
    "guerre de trente ans": "thirty years' war 1618 1648",
    "guerre de cent ans": "hundred years' war 1337 1453",
    "guerre de sept ans": "seven years war 1756 1763",
    "guerre": "war",
    "insurrection": "war",
    "siège de mayence": "siege of mainz",
    "siège": "siege",
    "commune": "commune",
    "défense": "battle",
    "révolution": "revolution"
}
```

Code source 6 – Table de conversion pour les évènements historiques

```
status = {
    "empereur": "",
    "impératrice": "",
    "général": "general",
    "reine": "queen",
    "roi": "king",
    "princesse": "princess",
    "prince": "prince",
    "archiduchesse": "",
    "archiduc": "",
    "duchesse": "duchess",
    "duc": "duke",
    "famille": "family",
    "seigneur": "",
    "vicomtesse": "",
    "victesse": "",
    "vicomte": "",
    "victe": "",
    "comtesse palatine": "countess palatine",
    "comtesse": "",
    "ctesse": "",
    "comte": "",
    "cte": "",
    "cardinal": "",
    "pape": "pope",
    "lord": "",
    "chevalier": "",
    "marquise": "",
    "marquis": "",
    "sire": "",
    "baronnesse": "",
    "baronne": "",
    "baron": "",
    "abbé": "",
    "madame": "",
    "mme": "",
    "monsieur": "",
    "mr": "",
    "docteur": "",
    "maréchale": "",
    "maréchal": "",
    "mademoiselle": "",
    "melle": "",
    "mlle": "",
    "sir": ""
}
```

Code source 7 – Liste de colonies pour la détection de motifs

```
comp_names = {
    "arm ch": "armand-charles",
    "ch m": "charles-marie",
    "ch l f": "charles-louis-françois",
    "f m": "francois-marie",
    "fr emm.": "françois-emmanuel",
    "j ant": "jean-antoine",
    "j f": "jean-francois",
    "j m": "jean-marie",
    "j j": "jean-jacques",
    "j l": "jean-louis",
    "j b": "jean-baptiste",
    "j p": "jean-pierre",
    "j pierre": "jean-pierre",
    "l f": "louis-françois",
    "m f": "marius-felix",
    "franc rené": "francois-rené",
    "m madeleine": "marie-madeleine",
    "ph h": "philippe henri",
    "p aug": "pierre auguste",
    "p alex": "pierre alexandre",
    "p j": "pierre-jean",
    "j sylvain": "jean-sylvain",
    "l ph": "louis-philippe",
    "edm ch": "edmond-charles",
    "ch marie": "charles-marie"
}
```

Code source 8 – Table de conversion permettant de remplacer un nom abrégé composé par sa version complète

```
names = {
    "ad": "adam",
    "alex": "alexandre",
    "alph": "alphonse",
    "ant": "antoine",
    "arm": "armand",
    "aug": "auguste",
    "ch": "charles",
    "cl": "claude",
    "dom": "dominique",
    "emm": "emmanuel",
    "ed": "edouard",
    "et": "etienne",
    "ét": "etienne",
    "ferd": "ferdinand",
    "fred": "frederic",
    "fr": "françois",
    "franc": "françois",
    "franç": "françois",
    "fréd": "frédéric",
    "g": "guillaume",
    "guill": "guillaume",
    "gab": "gabriel",
    "jh": "joseph",
    "jacq": "jacques",
    "jos": "joseph",
    "math": "matthieu",
    "nic": "nicolas",
    "ph": "philippe",
    "v": "victor",
    "vr": "victor",
}
```

Code source 9 – Table de conversion permettant de remplacer un nom abrégé non composé par sa version complète

```
def rgx_abvcomp(nstr):
    """
    try to extract an abbreviated composed first name. if there is no match, return None
    pattern
    -------
    the patterns in the example below are simplified to keep things readable
    - two strings separated by a "-" or "\s"
    - the first or second string can be a full name ([A-Z][a-z]+) or an abbreviation ([A-Z][a-z]*\.)
    - if the strings are separated by "\s", they must be finished by "\."
    (to be sure that we don't capture full names, i.e: "J. Ch."  can be captured,
    but not "Jean Charles")
    - complex names with 3 or more words must have "-" and at least one "\."
    - (\s|$) and (^|\s) are safeguards to avoid matching the end or beginning of another word
    examples
    --------
    matched : M.-Madeleine Pioche de la Vergne  # matched string : M.-Madeleine
    matched : C.-A. de Ferriol  # matched string : C.-A.
    matched : J. F.  # matched string : J. F.
    matched : Jean F.  # matched string : Jean F.
    matched : Jean-F.  # matched string : Jean-F.
    matched : A M  # matched string : A M
    matched : C.-Edm.-G.  # matched string : C.-Edm.-G.
    matched : Charles-Edm.-G.  # matched string : Charles-Edm.-G.
    not matched : Anne M
    not matched : Claude Henri blabla
    not matched : Claude Henri
    :param nstr: the name string used as input
    :return: the matched string if there is a match ; None if there is no match
```

```
    """
    mo = re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùû
    ↪  üøœæç]*"
           + "\.?-[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.(
           ↪  \s|,|$)", nstr)
           ↪  \
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôö
         ↪  úùûüøœæç]*\."
           + "-[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.?(\s
           ↪  |,|$)", nstr)
           ↪  \
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.?\s"
           + "[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.(\s|,
           ↪  |$)", nstr)
           ↪  \
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôö
         ↪  úùûüøœæç]*\.?"
           + "\s[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.(\s|,|$)", nstr) \
         or re.search(r"(^|,|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.?\s[A-ZÀÂÄÈÉÊËÏÔŒÙ
         ↪  ÛÜŸ]\.?(\s|,|$)", nstr)
         ↪  \
         or re.search(r"([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ]\.){2,}", nstr) \
         or re.search(r"(^|,|\s)([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóô
         ↪  öúùûüøœæç]*\.?-)+"
           + "([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.)(\s
           ↪  |,|$)", nstr)
           ↪  \
         or re.search(r"(^|,|\s)([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóô
         ↪  öúùûüøœæç]*\.-)+"
            + "([A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.?)
            ↪  (\s|,|$)",
            ↪  nstr)
    if mo is not None:
        return mo[0]
    else:
        return None
```

Code source 10 – Fonction permettant d'identifier et d'extraire un nom abrégé composé

```
def rgx_abvsimp(nstr):
    """
    try to extract a "simple" (not composed) abbreviated first name. if there is no match, return None
    pattern
    -------
    a capital letter (possibly followed by a certain number of lowercase letters)
    ended with a dot. (\s|$) and (^|\s) are safeguards to avoid matching the beginning
    end of another word.
    *warning* : it can also capture parts of composed abbreviated names => must be used
    in an if-elif after trying to match a composed abbreviated name
    examples
    --------
    matched : bonjour Ad.  # matched string : Ad.
    matched : J. baronne  # matched string : J.
    matched : J. F.  # matched string : J.
    matched : Jean F.  # matched string : F.
    not matched : A.-M.
    not matched : Anne M
    not matched : Hector
    :param nstr: the name string used as input
    :return:
    """
    mo = re.search(r"(^|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]*\.(\s|$|,)", nstr)
    if mo is not None:
        return mo[0]
    else:
        return None
```

Code source 11 – Fonction permettant de repérer et d'extraire un nom abrégé simple

```
def rgx_complnm(nstr):
    """
    try to extract a complete name from a string. if there is noœ
↪   match, return None
    pattern
    -------
    - an uppercase letter followed by several lowercase letters ;
    - this pattern can be repeated several times, separated by a space
↪   or "-"
    - (\s|$) and (^|\s) are safeguards to avoid matching the beginning
↪   or end of another word.
    :param nstr: the string from which a name should be extracted
    :return:
    """
    mo = re.search(r"(^|\s)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüø⌋
↪   œæç]+"
        + "((\s|-)[A-ZÀÂÄÈÉÊËÏÔŒÙÛÜŸ][a-zàáâäéèêëíìîïòóôöúùûüøœæç]+)*($⌋
↪   |\s|,)",
↪   nstr)
    if mo is not None:
        return mo[0]
    else:
        return None
```

Code source 12 – Fonction permettant d'identifier et d'extraire un nom complet non abrégé

```xml
<item n="264" xml:id="CAT_000327_e264">
    <!-- ... -->
    <name type="author">Spontini (Gaspard)</name>
    <trait>
        <p>célèbre compositeur, auteur de la Vestale, né en 1779, mort
        ↪ en 1851.</p>
    </trait>
    <!-- ... -->
</item>
```

(a) La source XML-TEI

```python
{
    'fname': 'gaspard ',
    'lname': 'spontini ',
    'nobname_sts': '',
    'status': '',
    'dates': '1779 1851 ',
    'function': 'writer',
    'rebuilt': False
}
```

(b) Le dictionnaire d'informations structurées

```
gaspard spontini 1779 1851 writer
gaspard spontini 1851 writer
gaspard spontini 1779 writer
gaspard spontini writer
spontini 1779 1851 writer
spontini 1851 writer
spontini 1779 writer
spontini writer
```

(c) Les recherches lancées sur l'API *Wikidata*

FIGURE 4 – De la TEI à l'API : document en entrée, informations extraites et chaînes de caractères recherchées

Ci dessus (4) sont présentés l'entrée et les données produites lors de l'alignement avec *Wikidata* : en premier les données en entrée, ensuite les informations qui en sont extraites et enfin les différentes chaînes de caractères recherchées. Il est à noter que, dans la plupart des cas, seules une ou deux recherches sont faites sur l'API avant d'obtenir un résultat ; ici, huit chaînes différentes ont été construites et recherchées, ce qui représente l'intégralité de l'algorithme.

```
{
    "head": {
        "vars": [
            "painting",
            "paintingLabel"
        ]
    },
    "results": {
        "bindings": [
            {
                "painting": {
                    "type": "uri",
                    "value": "http://www.wikidata.org/entity/Q4500028"
                },
                "paintingLabel": {
                    "xml:lang": "en",
                    "type": "literal",
                    "value": "Circular Dance"
                }
            },
            {
                "painting": {
                    "type": "uri",
                    "value": "http://www.wikidata.org/entity/Q19882576"
                },
                "paintingLabel": {
                    "xml:lang": "en",
                    "type": "literal",
                    "value": "Cats (rayist percep.[tion] in rose,
                    ↪  black, and yellow)"
                }
            },
            {
                "painting": {
                    "type": "uri",
                    "value": "http://www.wikidata.org/entity/Q19883614"
                },
```

```
                "paintingLabel": {
                    "xml:lang": "en",
                    "type": "literal",
                    "value": "Landscape 47"
                }
            },
            {
                "painting": {
                    "type": "uri",
                    "value": "http://www.wikidata.org/entity/Q19883648"
                },
                "paintingLabel": {
                    "xml:lang": "en",
                    "type": "literal",
                    "value": "Rayonism Blue-Green Forest"
                }
            },
            {
                "painting": {
                    "type": "uri",
                    "value": "http://www.wikidata.org/entity/Q20200781"
                },
                "paintingLabel": {
                    "xml:lang": "en",
                    "type": "literal",
                    "value": "Decor for the Ballet \"Liturgie\""
                }
            }
        ]
    }
}
```


Code source 13 – Exemple de réponse SPARQL au format JSON pour la requête 6.1

```
<?xml version="1.0" ?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

    <head>

        <variable name="painting"/>

        <variable name="paintingLabel"/>

    </head>

    <results>

        <result>

            <binding name="painting">

                <uri>http://www.wikidata.org/entity/Q4500028</uri>

            </binding>

            <binding name="paintingLabel">

                <literal xml:lang="en">Circular Dance</literal>

```

```
38          </binding>
39
40
41      </result>
42
43
44      <result>
45
46
47          <binding name="painting">
48
49
50              <uri>http://www.wikidata.org/entity/Q19882576</uri>
51
52
53          </binding>
54
55
56          <binding name="paintingLabel">
57
58
59              <literal xml:lang="en">Cats (rayist percep.[tion] in
                ↪  rose, black, and yellow)</literal>
60
61
62          </binding>
63
64
65      </result>
66
67
68      <result>
69
70
71          <binding name="painting">
72
73
74              <uri>http://www.wikidata.org/entity/Q19883614</uri>
```

```
75
76
77              </binding>
78
79
80              <binding name="paintingLabel">
81
82
83                  <literal xml:lang="en">Landscape 47</literal>
84
85
86              </binding>
87
88
89          </result>
90
91
92          <result>
93
94
95              <binding name="painting">
96
97
98                  <uri>http://www.wikidata.org/entity/Q19883648</uri>
99
100
101             </binding>
102
103
104             <binding name="paintingLabel">
105
106
107                 <literal xml:lang="en">Rayonism Blue-Green
                    ↪  Forest</literal>
108
109
110             </binding>
111
```

```
112
113         </result>
114
115
116         <result>
117
118
119             <binding name="painting">
120
121
122                 <uri>http://www.wikidata.org/entity/Q20200781</uri>
123
124
125             </binding>
126
127
128             <binding name="paintingLabel">
129
130
131                 <literal xml:lang="en">Decor for the Ballet
                    ↪  &quot;Liturgie&quot;</literal>
132
133
134             </binding>
135
136
137         </result>
138
139
140     </results>
141
142
143 </sparql>
```

Code source 14 – Exemple de réponse SPARQL au format XML pour la requête 6.1

```
{
    # entrées précédentes...
    "Q108438226": {
        "instance": [
            "Q3305213"
        ],
        "instanceL": [
            "painting"
        ],
        "gender": [],
        "genderL": [],
        "citizenship": [],
        "citizenshipL": [],
        "lang": [],
        "langL": [],
        "deathmanner": [],
        "deathmannerL": [],
        "birthplace": [],
        "birthplaceL": [],
        "deathplace": [],
        "deathplaceL": [],
        "residplace": [],
        "residplaceL": [],
        "burialplace": [],
        "burialplaceL": [],
        "educ": [],
        "educL": [],
        "religion": [],
        "religionL": [],
        "occupation": [],
        "occupationL": [],
        "award": [],
        "awardL": [],
        "position": [],
        "positionL": [],
        "member": [],
        "memberL": [],
```

```
        "nobility": [],
        "nobilityL": [],
        "workcount": [],
        "conflictcount": [],
        "img": [
            "http://commons.wikimedia.org/wiki/Special:FilePath/Louis%2
            ↪ 0de%20Frott%C3%A9.jpg"
        ],
        "signature": [],
        "birth": [],
        "death": [],
        "title": [
            "Louis de Frott\u00e9 (1755-1800), g\u00e9n\u00e9ral
            ↪ vend\u00e9en"
        ],
        "inception": [
            "1822-01-01"
        ],
        "author": [],
        "authorL": [],
        "pub": [],
        "pubL": [],
        "pubplace": [],
        "pubplaceL": [],
        "pubdate": [],
        "creator": [
            "Q51077254"
        ],
        "creatorL": [
            "Louise Bouteiller"
        ],
        "material": [
            "Q12321255",
            "Q296955"
        ],
        "materialL": [
        "canvas",
            "oil paint"
```

```
        ],
        "height": [
            "216"
        ],
        "genre": [
            "Q134307"
        ],
        "genreL": [
            "portrait"
        ],
        "movement": [],
        "movementL": [],
        "creaplace": [],
        "creaplaceL": [],
        "viafID": [],
        "bnfID": [],
        "isniID": [],
        "congressID": [],
        "idrefID": []
    },
    "Q3383960": {
        "instance": [
            "Q5"
        ],
        "instanceL": [
            "human"
        ],
        "gender": [
            "Q6581097"
        ],
        "genderL": [
            "male"
        ],
        "citizenship": [
            "Q142"
        ],
        "citizenshipL": [
            "France"
```

```
        ],
        "lang": [
            "Q150"
        ],
        "langL": [
            "French"
        ],
        "deathmanner": [],
        "deathmannerL": [],
        "birthplace": [
            "Q1148468"
        ],
        "birthplaceL": [
            "Vendargues"
        ],
        "deathplace": [],
        "deathplaceL": [],
        "residplace": [],
        "residplaceL": [],
        "burialplace": [],
        "burialplaceL": [],
        "educ": [],
        "educL": [],
        "religion": [],
        "religionL": [],
        "occupation": [
            "Q47064",
            "Q82955",
            "Q189290"
        ],
        "occupationL": [
            "military personnel",
            "politician",
            "military officer",
            "officer"
        ],
        "award": [
            "Q10855226",
```

```
            "Q11593374",
            "Q1313340"
        ],
        "awardL": [
            "Grand Croix of the L\u00e9gion d'honneur",
            "Knight of the Royal and Military Order of Saint Louis",
            "names inscribed under the Arc de Triomphe",
            "Grand Cross of the Legion of Honour"
        ],
        "position": [
            "Q21032625",
            "Q15904689"
        ],
        "positionL": [
            "member of the Chamber of Peers",
            "Governor of Algeria"
        ],
        "member": [],
        "memberL": [],
        "nobility": [
            "Q165503"
        ],
        "nobilityL": [
            "baron"
        ],
        "workcount": [],
        "conflictcount": [
            "1"
        ],
        "img": [
            "http://commons.wikimedia.org/wiki/Special:FilePath/Berthez
            ↪ %C3%A8ne.JPG"
        ],
        "signature": [],
        "birth": [
            "1775-03-24"
        ],
        "death": [
```

```
            "1847-10-09"
        ],
        "title": [],
        "inception": [],
        "author": [],
        "authorL": [],
        "pub": [],
        "pubL": [],
        "pubplace": [],
        "pubplaceL": [],
        "pubdate": [],
        "creator": [],
        "creatorL": [],
        "material": [],
        "materialL": [],
        "height": [],
        "genre": [],
        "genreL": [],
        "movement": [],
        "movementL": [],
        "creaplace": [],
        "creaplaceL": [],
        "viafID": [
            "15030322"
        ],
        "bnfID": [
            "14643064j"
        ],
        "isniID": [
            "0000 0000 4820 9162"
        ],
        "congressID": [
            "n2005062361"
        ],
        "idrefID": [
            "088623629"
        ]
    },
```

```
225     # entrées suivantes...
226 }
```

Code source 15 – Extrait de la base de connaissance constituée grâce à `SPARQL` suite à l'alignement avec des entités *Wikidata*

```
<listPrefixDef>
    <prefixDef
        ident="wd"
        matchPattern="(Q[0-9]+)"
        replacementPattern="https://www.wikidata.org/wiki/$1">
        <p>
            In the <gi>body</gi>, the <att>ref</att> attributes
            containted in <gi>name</gi> elements are pointing to to a
            <ref target="https://www.wikidata.org/wiki/">Wikidata</ref>
            identifier by using the <code>wd:</code> prefix. This
            ↪ <gi>prefixDef</gi>
            allows to rebuilt the complete URL from a wikidata
            ↪ identifier by
            replacing the <code>wd:</code> prefix with:
            <code>https://www.wikidata.org/wiki/</code>.
        </p>
    </prefixDef>
</listPrefixDef>
```

Code source 16 – Le `tei:listPrefixDef` décrivant le rôle du préfixe « wd » dans les catalogues

```
<?xml version='1.0' encoding='utf-8'?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>KatAPI query results</title>
        <respStmt>
          <resp>File created automatically by KatAPI, an API developped
          ↪ as part of the</resp>
          <orgName>
            <ref target="https://katabase.huma-num.fr/">MSS / Katabase
            ↪ project.</ref>
          </orgName>
        </respStmt>
        <respStmt>
          <resp>Production of the source data:</resp>
          <orgName>Projet e-ditiones</orgName>
          <orgName>Katabase</orgName>
          <orgName> Manuscript SaleS Catalogue</orgName>
        </respStmt>
        <funder>
          <orgName>Université de Neuchâtel</orgName>
          <orgName>Université de Genève</orgName>
          <orgName>École normale Supérieure</orgName>
        </funder>
      </titleStmt>
      <publicationStmt>
        <ab>
          <date type="file-creation-date" when-iso="2022-08-24T16:41:40
          ↪ .267860">2022-08-24T16:41:40.267860</date>
          <ref type="http-status-code" target="https://developer.mozill
          ↪ a.org/en-US/docs/Web/HTTP/Status/200">200</ref>
          <note>Original data made available under Creative Commons
          ↪ Attribution 2.0 Generic (CC BY 2.0)</note>
          <table>
            <head>Query parameters</head>
            <row>
```

```
33              <cell role="key" xml:id="level">level</cell>
34              <cell role="key" xml:id="id">id</cell>
35              <cell role="key" xml:id="format">format</cell>
36            </row>
37            <row>
38              <cell role="value" corresp="level">item</cell>
39              <cell role="value" corresp="id">CAT_000204_e108_d1</cell>
40              <cell role="value" corresp="format">tei</cell>
41            </row>
42          </table>
43        </ab>
44      </publicationStmt>
45      <sourceDesc>
46        <p>Sources may come from different documents. See the
          ↪ corresponding XML-TEI catalogues on
47          <ref target="https://github.com/katabase/Application/tree/mai
            ↪ n/APP/data">Github</ref>
48          for detailed description of the sources.
49        </p>
50      </sourceDesc>
51    </fileDesc>
52    <encodingDesc>
53      <listPrefixDef>
54        <prefixDef ident="wd" matchPattern="(Q[0-9]+)"
          ↪ replacementPattern="https://www.wikidata.org/wiki/$1">
55        </prefixDef>
56      </listPrefixDef>
57      <classDecl>
58        <taxonomy xml:id="document_format">
59          <desc>Document format</desc>
60          <category xml:id="document_format_1">
61            <catDesc>In-folio</catDesc>
62          </category>
63          <category xml:id="document_format_2">
64            <catDesc>In-2°</catDesc>
65          </category>
66          <category xml:id="document_format_3">
67            <catDesc>In-3°</catDesc>
```

```
68          </category>
69          <category xml:id="document_format_4">
70            <catDesc>In-quarto</catDesc>
71          </category>
72          <category xml:id="document_format_8">
73            <catDesc>In-octavo</catDesc>
74          </category>
75          <category xml:id="document_format_12">
76            <catDesc>In-12</catDesc>
77          </category>
78          <category xml:id="document_format_16">
79            <catDesc>In-16</catDesc>
80          </category>
81          <category xml:id="document_format_18">
82            <catDesc>In-18</catDesc>
83          </category>
84          <category xml:id="document_format_32">
85            <catDesc>In-32</catDesc>
86          </category>
87          <category xml:id="document_format_40">
88            <catDesc>In-40</catDesc>
89          </category>
90          <category xml:id="document_format_48">
91            <catDesc>In-48</catDesc>
92          </category>
93          <category xml:id="document_format_64">
94            <catDesc>In-64</catDesc>
95          </category>
96          <category xml:id="document_format_101">
97            <catDesc>In-folio oblong</catDesc>
98          </category>
99          <category xml:id="document_format_102">
100           <catDesc>In-2° oblong</catDesc>
101         </category>
102         <category xml:id="document_format_103">
103           <catDesc>In-3° oblong</catDesc>
104         </category>
105         <category xml:id="document_format_104">
```

```
106           <catDesc>In-quarto oblong</catDesc>
107         </category>
108         <category xml:id="document_format_108">
109           <catDesc>In-octavo oblong</catDesc>
110         </category>
111         <category xml:id="document_format_112">
112           <catDesc>In-12 oblong</catDesc>
113         </category>
114         <category xml:id="document_format_116">
115           <catDesc>In-16 oblong</catDesc>
116         </category>
117         <category xml:id="document_format_118">
118           <catDesc>In-18 oblong</catDesc>
119         </category>
120         <category xml:id="document_format_132">
121           <catDesc>In-32 oblong</catDesc>
122         </category>
123         <category xml:id="document_format_140">
124           <catDesc>In-40 oblong</catDesc>
125         </category>
126         <category xml:id="document_format_148">
127           <catDesc>In-48 oblong</catDesc>
128         </category>
129         <category xml:id="document_format_164">
130           <catDesc>In-64 oblong</catDesc>
131         </category>
132       </taxonomy>
133       <taxonomy xml:id="document_term">
134         <desc>Document types</desc>
135         <category xml:id="document_term_1">
136           <catDesc>Apostille autographe signée | signed autograph
              ↪ apostilla</catDesc>
137         </category>
138         <category xml:id="document_term_2">
139           <catDesc>Pièce autographe signée | signed autograph
              ↪ document</catDesc>
140         </category>
141         <category xml:id="document_term_3">
```

```
142           <catDesc>Pièce autographe | autograph document</catDesc>
143         </category>
144         <category xml:id="document_term_4">
145           <catDesc>Pièce signée | signed document</catDesc>
146         </category>
147         <category xml:id="document_term_5">
148           <catDesc>Billet autographe signé | signed autograph short
              ↪ document</catDesc>
149         </category>
150         <category xml:id="document_term_6">
151           <catDesc>Billet signé | signed short document</catDesc>
152         </category>
153         <category xml:id="document_term_7">
154           <catDesc>Lettre autographe signée | signed autograph
              ↪ letter</catDesc>
155         </category>
156         <category xml:id="document_term_8">
157           <catDesc>Lettre autographe | autograph letter</catDesc>
158         </category>
159         <category xml:id="document_term_9">
160           <catDesc>Lettre signée | signed letter</catDesc>
161         </category>
162         <category xml:id="document_term_10">
163           <catDesc>Brevet signé | signed certificate</catDesc>
164         </category>
165         <category xml:id="document_term_11">
166           <catDesc>Quittance autographe signée | signed autograph
              ↪ receipt</catDesc>
167         </category>
168         <category xml:id="document_term_12">
169           <catDesc>Quittance signée | signed receipt</catDesc>
170         </category>
171         <category xml:id="document_term_13">
172           <catDesc>Manuscrit autographe | autograph
              ↪ manuscript</catDesc>
173         </category>
174         <category xml:id="document_term_14">
175           <catDesc>Chanson autographe | autograph song</catDesc>
```

```
176         </category>
177         <category xml:id="document_term_15">
178           <catDesc>Document (?) Autographe signé | signed autograph
              ↪ document (?)</catDesc>
179         </category>
180       </taxonomy>
181     </classDecl>
182   </encodingDesc>
183 </teiHeader>
184 <text>
185   <body>
186     <div type="search-results">
187       <list><head>Search results</head><item n="108"
          ↪ xml:id="CAT_000204_e108">
188         <num type="lot">108</num>
189         <name type="author" ref="wd:Q255">BEETHOVEN (L. van)</name>
190         <trait>
191           <p>le grand compositeur de musique.</p>
192         </trait>
193         <desc xml:id="CAT_000204_e108_d1">
194           <term ana="#document_type_9">L. s.</term> à M. M.
              ↪ Schlesinger, à Berlin; Vienne, <date
              ↪ when="1820-05-31">31 mai 1820</date>, <measure
              ↪ type="length" unit="p" n="2">2 p.</measure>
195           <measure type="format" unit="f"
              ↪ ana="#document_format_4">in-4</measure>, cachet</desc>
196         <note>Curieuse lettre sur ses ouvrages. Il leur accorde le
            ↪ droit de vendre ses compositions en Angleterre, y compris
            ↪ les airs écossais, aux conditions indiquées par lui. Il
            ↪ s'engage à leur livrer dans trois mois trois sonates pour
            ↪ le prix de 90 florins qu'ils ont fixé. C'est pour leur
            ↪ être agréable qu'il accepte un si petit honoraire. « Je
            ↪ suis habitué à faire des sacrifices, la composition de
            ↪ mes OEuvres n'étant pas faite seulement au point de vue
            ↪ du rapport des honoraires, mais surtout dans l'intention
            ↪ d'en tirer quelque chose de bon pour l'art.»</note>
197       </item>
198       </list>
```

```
199       </div>
200     </body>
201   </text>
202 </TEI>
```

Code source 17 – Exemple de réponse de *KatAPI* au niveau `item` en `TEI`

```
{
    "head": {
        "encoding_desc": {
            "prefix_definition": {
                "wd:": "https://www.wikidata.org/wiki/"
            },
            "taxonomy_format": {
                "document_format_1": "In-folio",
                "document_format_101": "In-folio oblong",
                "document_format_102": "In-2\u00b0 oblong",
                "document_format_103": "In-3\u00b0 oblong",
                "document_format_104": "In-quarto oblong",
                "document_format_108": "In-octavo oblong",
                "document_format_112": "In-12 oblong",
                "document_format_116": "In-16 oblong",
                "document_format_118": "In-18 oblong",
                "document_format_12": "In-12",
                "document_format_132": "In-32 oblong",
                "document_format_140": "In-40 oblong",
                "document_format_148": "In-48 oblong",
                "document_format_16": "In-16",
                "document_format_164": "In-64 oblong",
                "document_format_18": "In-18",
                "document_format_2": "In-2\u00b0",
                "document_format_3": "In-3\u00b0",
                "document_format_32": "In-32",
                "document_format_4": "In-quarto",
                "document_format_40": "In-40",
                "document_format_48": "In-48",
                "document_format_64": "In-64",
                "document_format_8": "In-octavo"
            },
            "taxonomy_term": {
                "document_term_1": "Apostille autographe sign\u00e9e |
                ↪  signed autograph apostilla",
                "document_term_10": "Brevet sign\u00e9 | signed
                ↪  certificate",
```

```
                "document_term_11": "Quittance autographe sign\u00e9e |
                ↪  signed ",
                "document_term_12": "Quittance sign\u00e9e | signed
                ↪  receipt",
                "document_term_13": "Manuscrit autographe | autograph
                ↪  manuscript",
                "document_term_14": "Chanson autographe | autograph
                ↪  song",
                "document_term_15": "Document (?) Autographe sign\u00e9
                ↪  | signed autograph document (?)",
                "document_term_2": "Pi\u00e8ce autographe sign\u00e9e |
                ↪  signed autograph document",
                "document_term_3": "Pi\u00e8ce autographe | signed
                ↪  autograph",
                "document_term_4": "Pi\u00e8ce sign\u00e9e | signed
                ↪  document",
                "document_term_5": "Billet autographe sign\u00e9 |
                ↪  signed autograph short document",
                "document_term_6": "Billet sign\u00e9 | signed short
                ↪  document",
                "document_term_7": "Lettre autographe sign\u00e9e |
                ↪  signed autograph letter",
                "document_term_8": "Lettre autographe | autograph
                ↪  letter",
                "document_term_9": "Lettre sign\u00e9e | signed letter"
            }
        },
        "license": "Attribution 2.0 Generic (CC BY 2.0)",
        "query": {
            "format": "json",
            "level": "item",
            "name": "s\u00e9vign\u00e9",
            "orig_date": "1500-1800",
            "sell_date": "1800-1900"
        },
        "query_date": "2022-08-24T16:41:18.876602",
        "status_code": 200
    },
```

```
    "results": {
        "CAT_000185_e608_d1": {
            "author": "S\u00e9vign\u00e9",
            "author_wikidata_id": null,
            "date": "1685",
            "desc": "\n                        L. aut. \u00e0 sa fille;
            ↪  mercredi 14 f\u00e9vrier (1685), 7 p. 1/2\n
            ↪            in-4\n                    ",
            "format": 4,
            "number_of_pages": 7.5,
            "price": null,
            "sell_date": "1881",
            "term": 8
        },
        "CAT_000196_e179_d1": {
            "author": "S\u00c9VIGN\u00c9",
            "author_wikidata_id": null,
            "date": "1684",
            "desc": "\n                        L. aut. \u00e0 sa fille;
            ↪  Saumur, 18 septembre (1684), 3 p.\n
            ↪     in-4\n                    ",
            "format": 4,
            "number_of_pages": 3.0,
            "price": null,
            "sell_date": "1882",
            "term": 8
        },
        "CAT_000273_e234_d1": {
            "author": "S\u00c9VIGN\u00c9",
            "author_wikidata_id": null,
            "date": "1695-08-09",
            "desc": "\n                        L. a. s. (\u00e0
            ↪  Lamoignon); Nantes, 9 ao\u00fbt 1695, 3 p.\n
            ↪             in-4\n                    ",
            "format": 4,
            "number_of_pages": 3.0,
            "price": null,
            "sell_date": "1891",
```

```
94              "term": 7
95          }
96      }
97  }
```

Code source 18 – Exemple de réponse de *KatAPI* au niveau `item` en `JSON`

```xml
<?xml version='1.0' encoding='utf-8'?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>KatAPI query results</title>
        <respStmt>
          <resp>File created automatically by KatAPI, an API developped as part of the</resp>
          <orgName>
            <ref target="https://katabase.huma-num.fr/">MSS / Katabase project.</ref>
          </orgName>
        </respStmt>
        <respStmt>
          <resp>Production of the source data:</resp>
          <orgName>Projet e-ditiones</orgName>
          <orgName>Katabase</orgName>
          <orgName> Manuscript SaleS Catalogue</orgName>
        </respStmt>
        <funder>
          <orgName>Université de Neuchâtel</orgName>
          <orgName>Université de Genève</orgName>
          <orgName>École normale Supérieure</orgName>
        </funder>
      </titleStmt>
      <publicationStmt>
        <ab>
          <date type="file-creation-date" when-iso="2022-08-24T16:41:49.888578">2022-08-24T16:41:49.888578</date>
          <ref type="http-status-code" target="https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/200">200</ref>
          <note>Original data made available under Creative Commons Attribution 2.0 Generic (CC BY 2.0)</note>
          <table>
            <head>Query parameters</head>
            <row>
```

```
            <cell role="key" xml:id="level">level</cell>
            <cell role="key" xml:id="id">id</cell>
            <cell role="key" xml:id="format">format</cell>
          </row>
          <row>
            <cell role="value" corresp="level">cat_stat</cell>
            <cell role="value" corresp="id">CAT_000362</cell>
            <cell role="value" corresp="format">tei</cell>
          </row>
        </table>
      </ab>
    </publicationStmt>
    <sourceDesc>
      <p>Sources may come from different documents. See the
      ↪ corresponding XML-TEI catalogues on
        <ref target="https://github.com/katabase/Application/tree/mai
        ↪ n/APP/data">Github</ref>
        for detailed description of the sources.
      </p>
    </sourceDesc>
  </fileDesc>
  <encodingDesc>
    <classDecl>
      <taxonomy xml:id="cat_stat_keys">
        <desc>Description of the possible values for the
        ↪ <att>key</att> attribute in the body.</desc>
        <category xml:id="cat_stat_keys_sell_date">
          <catDesc>The year of the sale presented in the
          ↪ catalogue.</catDesc>
        </category>
        <category xml:id="cat_stat_keys_title">
          <catDesc>The title of the catalogue, and, by extension, of
          ↪ the sale.</catDesc>
        </category>
        <category xml:id="cat_stat_keys_cat_type">
          <catDesc>The type of catalogue
          ↪ (`LAC`|`RDA`|`AUC`|`OTH`).</catDesc>
        </category>
```

```
65          <category xml:id="cat_stat_keys_currency">
66            <catDesc>The currency of the prices returned. `FRF`
              ↪ corresponds to french francs.</catDesc>
67          </category>
68          <category xml:id="cat_stat_keys_high_price_c">
69            <catDesc>The most expensive items in the catalogue, in
              ↪ constant 1900 francs.
70              In the whole API, the `_c` suffix indicates that a price
71              is expressed in constant 1900 francs.</catDesc>
72          </category>
73          <category xml:id="cat_stat_keys_high_price_items_c">
74            <catDesc>The most expensive items in the catalogue. Each
              ↪ item's
75              `@xml:id` is associated to its price, in constant
                ↪ francs.</catDesc>
76          </category>
77          <category xml:id="cat_stat_keys_item_count">
78            <catDesc>The number of manuscripts for sale in the
              ↪ catalogue.</catDesc>
79          </category>
80          <category xml:id="cat_stat_keys_low_price_c">
81            <catDesc>The least expensive item's price, in constant 1900
              ↪ francs.</catDesc>
82          </category>
83          <category xml:id="cat_stat_keys_mean_price_c">
84            <catDesc>The average price for a catalogue item, in
              ↪ constant 1900 francs.</catDesc>
85          </category>
86          <category xml:id="cat_stat_keys_median_price_c">
87            <catDesc>The median price for a catalogue item, in constant
              ↪ francs.</catDesc>
88          </category>
89          <category xml:id="cat_stat_keys_mode_price_c">
90            <catDesc>The mode price (the price that appears the most
              ↪ often in a catalogue)
91              for an item, in constant francs.</catDesc>
92          </category>
93          <category xml:id="cat_stat_keys_total_price_c">
```

```
94            <catDesc>The sum of each item's price, in constant
               ↪ francs</catDesc>
95          </category>
96          <category xml:id="cat_stat_keys_variance_price_c">
97            <catDesc>The variance of the prices inside the
               ↪ catalogue.</catDesc>
98          </category>
99        </taxonomy>
100     </classDecl>
101   </encodingDesc>
102 </teiHeader>
103 <text>
104   <body>
105     <div type="search-results">
106       <list>
107         <head>Search results</head>
108         <item ana="CAT_000362">
109           <label>CAT_000362</label>
110           <term key="title">Vente Jacques Charavay, août 1875, nº
               ↪ 185</term>
111           <term key="cat_type">LAC</term>
112           <term key="sell_date" type="date">1875</term>
113           <term key="item_count">106</term>
114           <term key="currency">FRF</term>
115           <term key="total_price_c" type="constant-price">1039</term>
116           <term key="low_price_c" type="constant-price">1.27</term>
117           <term key="high_price_c" type="constant-price">102.0</term>
118           <term key="mean_price_c"
               ↪ type="constant-price">9.810188679245282</term>
119           <term key="median_price_c" type="constant-price">4.59</term>
120           <term key="mode_price_c" type="constant-price">3.06</term>
121           <term key="variance_price_c"
               ↪ type="constant-price">194.2879773228907</term>
122           <term key="high_price_items_c" type="constant-price"
               ↪ ana="CAT_000362_e27096">102.0</term>
123         </item>
124       </list>
125     </div>
```

```
126     </body>
127   </text>
128 </TEI>
```

Code source 19 – Exemple de réponse de *KatAPI* au niveau `cat_stat` en `TEI`

```
{
    "head": {
        "encoding_desc": {
            "cat_type": "The type of catalogue
            ↪  (`LAC`|`RDA`|`AUC`|`OTH`)",
            "currency": "The currency of the prices returned. `FRF`
            ↪  corresponds to french francs.",
            "high_price_c": "The most expensive items in the catalogue,
            ↪  in constant 1900 francs. In the whole API, the `_c`
            ↪  suffix indicates that a price is expressed in constant
            ↪  1900 francs.",
            "high_price_items_c": "The most expensive items in the
            ↪  catalogue. Each item's `@xml:id` is associated to its
            ↪  price, in constant francs.",
            "item_count": "The number of manuscripts for sale in the
            ↪  catalogue.",
            "low_price_c": "The least expensive item's price, in
            ↪  constant 1900 francs.",
            "mean_price_c": "The average price for a catalogue item, in
            ↪  constant 1900 francs.",
            "median_price_c": "The median price for a catalogue item,
            ↪  in constant francs.",
            "mode_price_c": "The mode price (the price that appears the
            ↪  most often in a catalogue)For an item, in constant
            ↪  francs.",
            "sell_date": "The year of the sale presented in the
            ↪  catalogue.",
            "title": "The title of the catalogue, and, by extension, of
            ↪  the sale.",
            "total_price_c": "The sum of each item's price, in constant
            ↪  francs",
            "variance_price_c": "The variance of the prices inside the
            ↪  catalogue."
        },
        "license": "Attribution 2.0 Generic (CC BY 2.0)",
        "query": {
            "format": "json",
```

```
            "level": "cat_stat",
            "name": "RDA",
            "sell_date": "1800-1900"
        },
        "query_date": "2022-08-24T16:41:42.604421",
        "status_code": 200
    },
    "results": {
        "CAT_000001": {
            "cat_type": "RDA",
            "item_count": 379,
            "sell_date": "1845",
            "title": "Vente Charon, 14 mai 1845"
        },
        "CAT_000002": {
            "cat_type": "RDA",
            "currency": "FRF",
            "high_price_c": 51.0,
            "high_price_items_c": {
                "CAT_000002_e189": 51.0
            },
            "item_count": 189,
            "low_price_c": 1.53,
            "mean_price_c": 5.4235483870967744,
            "median_price_c": 3.57,
            "mode_price_c": 3.06,
            "sell_date": "Septembre - Decembre 1871",
            "title": "RDA, N\u00b029 (septembre 1871)",
            "total_price_c": 1008,
            "variance_price_c": 44.285716441207086
        },
        "CAT_000004": {
            "cat_type": "RDA",
            "item_count": 133,
            "sell_date": "1896",
            "title": "Vente Etienne Charavay, 30 mai 1986"
        },
        "CAT_000005": {
```

```
            "cat_type": "RDA",
            "currency": "FRF",
            "high_price_c": 663.0,
            "high_price_items_c": {
                "CAT_000005_e237": 663.0
            },
            "item_count": 137,
            "low_price_c": 2.04,
            "mean_price_c": 11.164651162790697,
            "median_price_c": 3.57,
            "mode_price_c": 2.55,
            "sell_date": "1856-10",
            "title": "Catalogue Laverdet, N\u00b03 (octobre 1856)",
            "total_price_c": 1440,
            "variance_price_c": 3410.2974357310254
        },
        "CAT_000017": {
            "cat_type": "RDA",
            "currency": "FRF",
            "high_price_c": 165.75,
            "high_price_items_c": {
                "CAT_000017_e126": 165.75
            },
            "item_count": 1,
            "low_price_c": 165.75,
            "mean_price_c": 165.75,
            "median_price_c": 165.75,
            "mode_price_c": 165.75,
            "sell_date": "1871",
            "title": "Collection Merlin, 31 octobre 1871",
            "total_price_c": 165,
            "variance_price_c": 0.0
        },
        "CAT_000018": {
            "cat_type": "RDA",
            "currency": "FRF",
            "high_price_c": 561.0,
            "high_price_items_c": {
```

```
                "CAT_000018_e144": 561.0
            },
            "item_count": 192,
            "low_price_c": 1.53,
            "mean_price_c": 10.565811518324606,
            "median_price_c": 5.1,
            "mode_price_c": 3.06,
            "sell_date": "1873-08",
            "title": "RDA, N\u00b037 (ao\u00fbt 1873)",
            "total_price_c": 2018,
            "variance_price_c": 1658.1071772210191
        },
    }
}
```


Code source 20 – Exemple de réponse de *KatAPI* au niveau `cat_stat` en `JSON`

```
<?xml version='1.0' encoding='utf-8'?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>KatAPI query results</title>
        <respStmt>
          <resp>File created automatically by KatAPI, an API developped as part of the</resp>
          <orgName>
            <ref target="https://katabase.huma-num.fr/">MSS / Katabase project.</ref>
          </orgName>
        </respStmt>
        <respStmt>
          <resp>Production of the source data:</resp>
          <orgName>Projet e-ditiones</orgName>
          <orgName>Katabase</orgName>
          <orgName> Manuscript SaleS Catalogue</orgName>
        </respStmt>
        <funder>
          <orgName>Université de Neuchâtel</orgName>
          <orgName>Université de Genève</orgName>
          <orgName>École normale Supérieure</orgName>
        </funder>
      </titleStmt>
      <publicationStmt>
        <ab>
          <date type="file-creation-date" when-iso="2022-08-24T16:41:16.346735">2022-08-24T16:41:16.346735</date>
          <ref type="http-status-code" target="https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/422">422</ref>
          <note>Original data made available under Creative Commons Attribution 2.0 Generic (CC BY 2.0)</note>
          <table>
            <head>Query parameters</head>
            <row>
```

```
33              <cell role="key" xml:id="format">format</cell>
34              <cell role="key" xml:id="sell_date">sell_date</cell>
35            </row>
36            <row>
37              <cell role="value" corresp="format">tei</cell>
38              <cell role="value" corresp="sell_date">2000?5000</cell>
39            </row>
40          </table>
41        </ab>
42      </publicationStmt>
43      <sourceDesc>
44        <p>Sources may come from different documents. See the
          ↪ corresponding XML-TEI catalogues on
45          <ref target="https://github.com/katabase/Application/tree/mai
            ↪ n/APP/data">Github</ref>
46          for detailed description of the sources.
47        </p>
48      </sourceDesc>
49    </fileDesc>
50  </teiHeader>
51  <text>
52    <body>
53      <div type="error-message">
54        <list>
55          <head>Invalid parameters or parameters combination</head>
56          <item ana="no_name+id">
57            <label>no_name+id</label>
58            <desc>You must specify at least a name or an id</desc>
59          </item>
60          <item ana="sell_date" corresp="sell_date">
61            <label>sell_date</label>
62            <desc>The format must match: \d{4}(-\d{4})?</desc>
63          </item>
64        </list>
65      </div></body>
66  </text>
67</TEI>
```

Code source 21 – Exemple de message d'erreur reçu par *KatAPI* en `TEI`

```
{
    "head": {
        "license": "Attribution 2.0 Generic (CC BY 2.0)",
        "query": {
            "api": "Durga Mahishasuraparini",
            "format": "json",
            "id": "CAT_0001_e0001_d0001",
            "name": "Guyan Yin",
            "sell_date": "200000"
        },
        "query_date": "2022-08-24T16:41:16.339770",
        "status_code": 422
    },
    "results": {
        "__error_type__": "Invalid parameters or parameters
        ↪  combination",
        "error_description": {
            "format": "The format must match: (tei|json)",
            "id_incompatible_params": "Invalid parameters with
            ↪  parameter id: ['sell_date']",
            "name+id": "You cannot provide both a name and an id",
            "sell_date": "The format must match: \\d{4}(-\\d{4})?",
            "unallowed_params": "Unallowed parameters for the API:
            ↪  ['api']"
        }
    }
}
```

Code source 22 – Exemple de message d'erreur reçu par *KatAPI* en `JSON`

Images

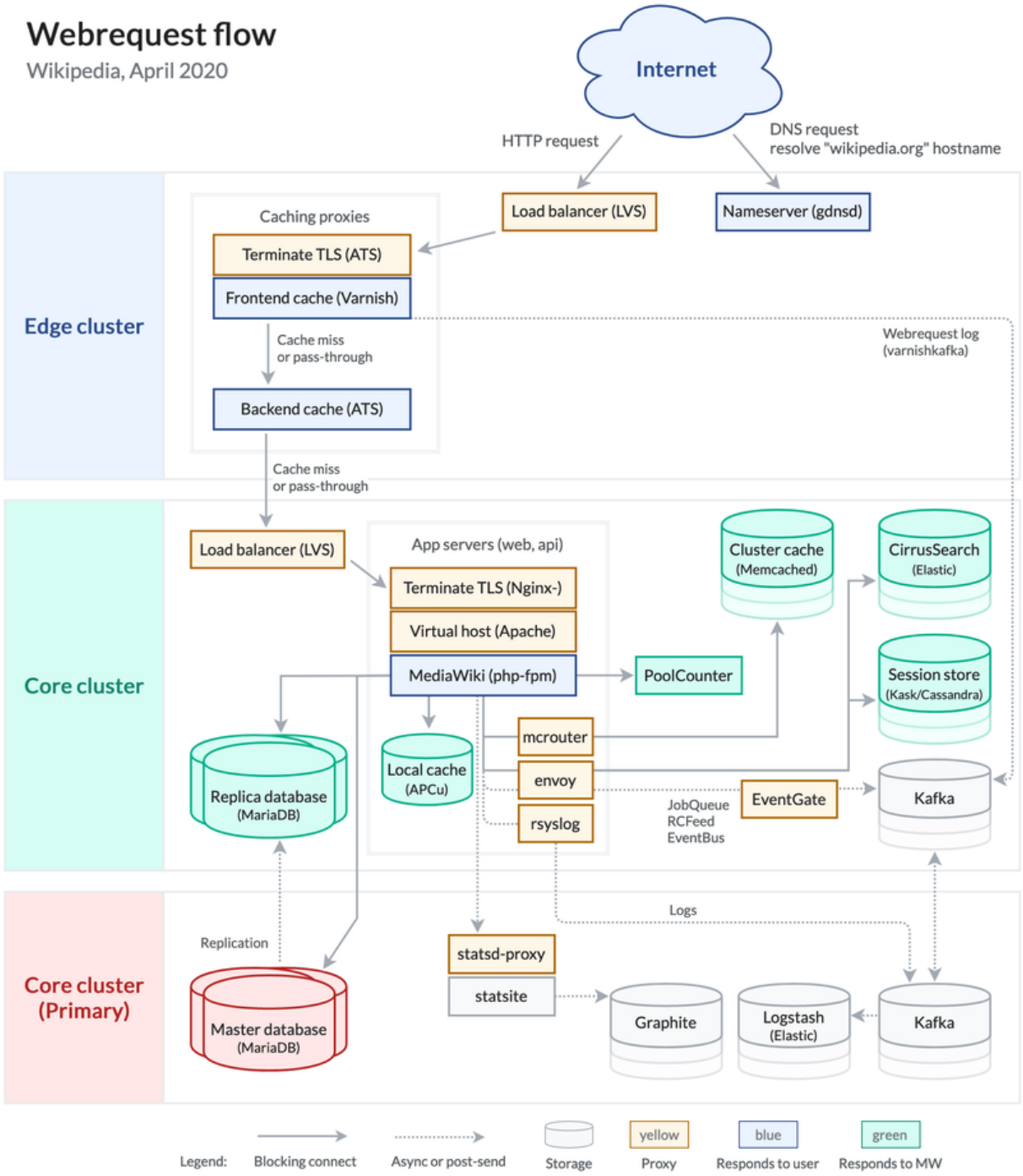


FIGURE 5 – L'architecture des bases de données de la *Wikimedia Foundation* (schéma réalisé par Timo Tijhof en 2020, disponible dans le domaine public avec la licence Creative Commons CC0.1.0 Universal Public Domain License)

Bibliographie

Projet *MSS / Katabase*

Corbières (Caroline), *Du catalogue au fichier TEI. Création d'un workflow pour encoder automatiquement en XML-TEI des catalogues d'exposition.* Mémoire pour le diplôme de master "Technologies numériques appliquées à l'histoire", Paris, École nationale des Chartes, 2020, URL : https://github.com/carolinecorbieres/Memoire_TNAH (visité le 13/06/2022).

Gabay (Simon), Rondeau du Noyer (Lucie) et Khemakhem (Mohamed), « Selling autograph manuscripts in 19th c. Paris : digitising the Revue des Autographes », dans *IX Convegno AIUCD*, Milan, 2020, URL : https://hal.archives-ouvertes.fr/hal-02388407 (visité le 13/06/2022).

Gabay (Simon), Rondeau du Noyer (Lucie), Gille Levenson (Matthias), Petkovic (Ljudmila) et Bartz (Alexandre), « Quantifying the Unknown : How many manuscripts of the marquise de Sévigné still exist ? », dans *Digital Humanities DH2020*, Ottawa, 2020, URL : https://hal.archives-ouvertes.fr/hal-02898929/ (visité le 13/06/2022).

Janès (Juliette), *Du catalogue papier au numérique. Une chaîne de traitement ouverte pour l'extraction d'information issue de documents structurés*, Mémoire pour le diplôme de master "Technologies numériques appliquées à l'histoire", Paris, École nationale des Chartes, 2021, URL : https://github.com/Juliettejns/Memoire_TNAH (visité le 13/06/2022).

Khemakhem (Mohamed), Romary (Laurent), Gabay (Simon), Bohbot (Hervé), Frontini (Francesca) et Luxardo (Giancarlo), « Automatically Encoding Encyclopedic-like Resources in TEI », dans *The annual TEI Conference and Members Meeting*, Tokyo, 2018, URL : https://hal.archives-ouvertes.fr/hal-01819505 (visité le 13/06/2022).

— « Information Extraction Workflow for Digitised Entry-based Documents. » Dans *DARIAH Annual event 2020*, Zagreb, 2020, URL : https://hal.archives-ouvertes.fr/hal-02508549 (visité le 13/06/1997).

Rondeau du Noyer (Lucie), *Encoder automatiquement des catalogues en XML-TEI. Principes, évaluation et application à la revue des autographes de la librairie Cha-*

ravay, Mémoire pour le diplôme de master "Technologies numériques appliquées à l'histoire", Paris, École nationale des Chartes, 2019, URL : https://github.com/lairaines/M2TNAH (visité le 13/06/2022).

Rondeau du Noyer (Lucie), Gabay (Simon), Khemakhem (Mohamed) et Romary (Laurent), « Scaling up Automatic Structuring of Manuscript Sales Catalogues », dans *TEI 2019 : What is text, really ? TEI and beyond*, Graz, 2019, URL : https://hal.archives-ouvertes.fr/hal-02272962 (visité le 13/06/2022).

Édition numérique, traitement automatisé et analyse de texte

Agirre (Eneko), Barrena (Ander), Lacalle (Oier Lopez de), Soroa (Aitor), Fernando (Samuel) et Stevenson (Mark), « Matching Cultural Heritage items to Wikipedia », dans *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, dir. Nicoletta Calzolari (Conference Chair), *et al.*, Istanbul, Turkey, 2012.

Bleeker (Elli), Haentjens Dekker (Ronald) et Buitendijk (Bram), « Texts as Hypergraphs : An Intuitive Representation of Interpretations of Text », *Journal of the Text Encoding Initiative*, 14 (2021), DOI : https://doi.org/10.4000/jtei.3919.

Boeglin (Noémie), Depeyre (Michel), Joliveau (Thierry) et Le Lay (Yves-François), « Pour une cartographie romanesque de Paris au XIXe siècle. Proposition méthodologique », dans *Conférence Spatial Analysis and GEOmatics*, Nice, France, 2016 (Actes de la conférence SAGEO'2016 - Spatial Analysis and GEOmatics), URL : https://hal.archives-ouvertes.fr/hal-01619600.

Brando (Carmen), Abadie (Nathalie) et Frontini (Fontini), « Évaluation de la qualité des sources du web de données pour la résolution d'entités nommées », *Ingénierie des systèmes d'information*, 21–5 (28 déc. 2016), p. 31-54, DOI : 10.3166/isi.21.5-6.31-54.

Brando (Carmen), Frontini (Francesca) et Ganascia (Jean-Gabriel), « REDEN : Named Entity Linking in Digital Literary Editions Using Linked Data Sets », *Complex Systems Informatics and Modeling Quarterly*–7 (29 juil. 2016), p. 60-80, DOI : 10.7250/csimq.2016-7.04.

Burnard (Lou), « How modeling standards evolve », dans *The shape of data in the digital humanities : modeling texts and text-based resources*, dir. Julia Flanders et Fotis Jannidis, London ; New York, 2019 (Digital research in the arts and humanities), p. 99-117.

— « What is TEI conformance, and why should you care ? », *Journal of the Text Encoding Initiative*, 12 (2019), DOI : https://doi.org/10.4000/jtei.1777.

Burnard (Lou), Schöch (Christian) et Odebrecht (Carolin), « In search of comity : TEI for distant reading », *Journal of the Text Encoding Initiative*, 14 (2021), doi : https://doi.org/10.4000/jtei.3500.

Chagué (Alix) et Clérice (Thibault), « Sharing HTR datasets with standardized metadata : the HTR-United initiative », dans *Documents anciens et reconnaissance automatique des écritures manuscrites*, Paris, France, 2022, url : https://hal.inria.fr/hal-03703989.

Christensen (Kelly), Gabay (Simon), Pinche (Ariane) et Camps (Jean-Baptiste), « SegmOnto – A Controlled Vocabulary to Describe Historical Textual Sources », dans *Documents anciens et reconnaissance automatique des écritures manuscrites*, Paris : École nationale des Chartes, 2022.

DeRose (Steven), Durand (David), Mylonas (Elli) et Renear (Allen), « What is Text, Really ? », *Journal of Computing in Higher Education*, I–2 (1990), p. 3-26, url : https://cs.brown.edu/courses/cs195-1/reading/WhatIsTextReally.pdf (visité le 26/08/2022).

Expression régulière, Wikipedia. L'encyclopédie libre, 2022, url : https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re (visité le 10/07/2022).

Frontini (Francesca), Brando (Carmen), Riguet (Marine), Jacquot (Clémence) et Jolivet (Vincent), « Annotation of Toponyms in TEI Digital Literary Editions and Linking to the Web of Data », *Matlit Revista do Programa de Doutoramento em Materialidades da Literatura*, 4–2 (11 juil. 2016), p. 49-75, doi : 10.14195/2182-8830_4-2_3.

Gabay (Simon), Camps (Jean-Baptiste), Pinche (Ariane) et Jahan (Claire), « SegmOnto : common vocabulary and practices for analysing the layout of manuscripts (and more) », dans *1st International Workshop on Computational Paleography (IWCP@ICDAR 2021)*, Lausanne, 2021, url : https://hal.archives-ouvertes.fr/hal-03336528 (visité le 29/06/2022).

Jannidis (Fotis) et Flanders (Julia), « Data modelling in a digital humanities context », dans *The shape of data in the digital humanities : modeling texts and text-based resources*, dir. Julia Flanders et Fotis Jannidis, London ; New York, 2019 (Digital research in the arts and humanities), p. 3-25.

Mendes (Pablo N.), Jakob (Max), García-Silva (Andrés) et Bizer (Christian), « DBpedia spotlight : shedding light on the web of documents », dans *Proceedings of the 7th International Conference on Semantic Systems - I-Semantics '11*, Graz, Austria, 2011, p. 1-8, doi : 10.1145/2063518.2063519.

Renear (Allen), Mylonas (Elli) et Durand (David), « Refining our Notion of What Text Really Is : The Problem of Overlapping Hierarchies », dans *Research in Humanities Computing*, dir. Nancy Ide et Susan Hockey, Oxford, 1996, url : https:

//cds.library.brown.edu/resources/stg/monographs/ohco.html (visité le 22/08/2022).

Ruiz Fabo (Pablo) et Poibeau (Thierry), « Mapping the Bentham Corpus : Concept-based Navigation », *Journal of Data Mining & Digital Humanities*, Atelier Digit_Hum (Data deluge : which skills for...[2019]), p. 5044, doi : 10.46298/jdmdh.5044.

Sahle (Patrick), « Digital modelling. Modelling the digital edition », dans *Medieval and modern manuscript studies in the digital age*, London/Cambridge, 2016, url : https://dixit.uni-koeln.de/wp-content/uploads/2015/04/Camp1-Patrick_Sahle_-_Digital_Modelling.pdf.

— « What is a scholarly digital edition ? », dans *Digital scholarly editing. Theories and practices*, dir. Matthew James Driscoll et Elena Pierazzo, Cambridge, 2016, url : http://books.openedition.org/obp/3397 (visité le 10/07/2022).

Sayers (Jentery), *Minimal definitions*, Minimal Computing - a working group of GO ::DH, 2 oct. 2016, url : http://go-dh.github.io/mincomp/thoughts/2016/10/02/minimal-definitions/ (visité le 10/06/2022).

Schweyer (Anne-Valérie), Burie (Jean-Christophe) et Nguyen (Tien Nam), « Analyse, Reconnaissance et Indexation des manuscrits CHAM », dans *Documents anciens et reconnaissance automatique des écritures manuscrites*, Paris, 2022.

Soudani (Aicha), Meherzi (Yosra), Bouhafs (Asma), Frontini (Francesca), Brando (Carmen), Dupont (Yoann) et Mélanie-Becquet (Frédérique), « Adaptation et évaluation de systèmes de reconnaissance et de résolution des entités nommées pour le cas de textes littéraires français du 19ème siècle », dans *Atelier Humanités Numériques Spatialisées (HumaNS'2018)*, Montpellier, 2018, url : https://hal.archives-ouvertes.fr/hal-01925816 (visité le 10/08/2022).

Stokes (Peter A.), Kiessling (Benjamin), Stökl Ben Ezra (Daniel), Tissot (Robin) et Gargem (Hassane), « The eScriptorium VRE for Manuscript Cultures », *Classics@ Journal*, 18 (2021), url : https://classics-at.chs.harvard.edu/classics18-stokes-kiessling-stokl-ben-ezra-tissot-gargem/ (visité le 14/07/2022).

Strubell (Emma), Ganesh (Ananya) et McCallum (Andrew), « Energy and Policy Considerations for Deep Learning in NLP », *CoRR*, abs/1906.02243 (2019), doi : https://doi.org/10.48550/arXiv.1906.02243.

TEI Consortium, *P5 : Guidelines for Electronic Text Encoding and Interchange*, Text Encoding Initiative, Version 4.4.0, 2022, url : https://tei-c.org/release/doc/tei-p5-doc/en/html/index.html.

Technologies du web : généralités

Boeckhout (Martin), Zielhuis (Gerhard A.) et Bredenoord (Annelien L.), « The FAIR guiding principles for data stewardship : fair enough ? », *European Journal of Human Genetics*, 26–7 (juil. 2018), p. 931-936, doi : 10.1038/s41431-018-0160-0.

Covington (Paul), Adams (Jay) et Sargin (Erme), « Deep Neural Networks for YouTube Recommendations », dans *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, 2016, url : https://research.google/pubs/pub45530/ (visité le 10/06/2022).

Fielding (Roy), Nottingham (Mark) et Reschke (Jilian), *HTTP Semantics. Request for Comments : 9110*, HTTP, juin 2022, url : https://httpwg.org/specs/rfc9110.html.

Mikolov (Tomas), Chen (Kai), Corrado (Greg) et Dean (Jeffrey), « Efficient Estimation of Word Representations in Vector Space » (, 2013), Publisher : arXiv Version Number : 3, doi : 10.48550/ARXIV.1301.3781.

Moteur de recherche, Wikipedia. L'encyclopédie libre, 2022, url : https://fr.wikipedia.org/wiki/Moteur_de_recherche (visité le 17/07/2022).

Namespace, Wikipedia. L'encyclopédie libre, 2022, url : https://en.wikipedia.org/wiki/Namespace (visité le 05/05/2022).

Wikimedia Foundation§Technology, Wikipedia. L'encyclopédie libre, 2022, url : https://en.wikipedia.org/wiki/Wikimedia_Foundation#Technology (visité le 01/08/2022).

Technologies du web : Web sémantique

Beckett (Dave), Broekstra (Jeen) et Hawke (Sandro), *SPARQL Query Results XML Format (Second Edition). W3C Recommendation 21 March 2013*, W3C, 21 mars 2013, url : https://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/ (visité le 10/05/2022).

Erxleben (Fredo), Günther (Michael), Krötzsch (Markus), Mendez (Julian) et Vrandečić (Denny), « Introducing Wikidata to the Linked Data Web », dans *The Semantic Web – ISWC 2014*, dir. Peter Mika, *et al.*, Series Title : Lecture Notes in Computer Science, Cham, 2014, t. 8796, p. 50-65, doi : 10.1007/978-3-319-11964-9_4.

Graph database, Wikipedia. L'encyclopédie libre, 2022, url : https://en.wikipedia.org/wiki/Graph_database (visité le 12/06/2022).

Harris (Steve), Seaborne (Andy) et Prud'hommeaux (Eric), *SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013*, W3C, 2013, url : http://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (visité le 10/06/2022).

Müller-Birn (Claudia), Karran (Benjamin), Lehmann (Janette) et Luczak-Rösch (Markus), « Peer-production system or collaborative ontology engineering effort : what is Wikidata ? », dans *Proceedings of the 11th International Symposium on Open Collaboration*, San Francisco California, 2015, p. 1-10, DOI : 10.1145/2788993.2789836.

Ontology (information science), Wikipedia. L'encyclopédie libre, 2022, URL : https://en.wikipedia.org/wiki/Ontology_(information_science) (visité le 01/02/2022).

Query optimization, Wikidata, 2022, URL : https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/query_optimization (visité le 06/06/2022).

Technologies du web : API

Almas (Bridget) et Clérice (Thibault), « Continuous Integration and Unit Testing of Digital Editions », *Digital Humanities Quarterly*, 11–4 (févr. 2018), Publisher : Alliance of Digital Humanities, URL : https://hal.archives-ouvertes.fr/hal-01709868 (visité le 12/07/2022).

Almas (Bridget), Clérice (Thibault), Cayless (Hugh), Jolivet (Vincent), Liuzzo (Pietro Maria), Romanello (Matteo), Robie (Jonathan) et Scott (Ian W.), *Distributed Text Services (DTS) : a Community-built API to Publish and Consume Text Collections as Linked Data*, mars 2021, URL : https://hal.archives-ouvertes.fr/hal-03183886 (visité le 01/01/2022).

API :Main page, MediaWiki, 2022, URL : https://www.mediawiki.org/wiki/API:Main_page (visité le 10/06/2022).

Clérice (Thibault), Almas (Bridget), Cayless (Hugh), Jolivet (Vincent), Morlock (Emmanuelle), Robie (Jonathan), Tauber (James), Witt (Jeffrey) et Liuzzo (Pietro), « From File Interoperability to Service Interoperability : The Distributed Text Services », dans *TEI 2018*, Tokyo, Japan, 2018, URL : https://hal.archives-ouvertes.fr/hal-02196659 (visité le 09/08/2022).

Edmond (Jennifer) et Garnett (Vicky), « APIs and Researchers : The Emperor's New Clothes ? », *International Journal of Digital Curation*, 10–1 (21 mai 2015), p. 287-297, DOI : 10.2218/ijdc.v10i1.369.

Fielding (Roy), *Architectural Styles and the Design of Network-based Software Architectures*, Version : PDF edition, 1-column for viewing online, Thèse de doctorat, Irvine, University of California, 2000, URL : https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (visité le 05/08/2022).

Goyet (Samuel), *De briques et de blocs. La fonction éditoriale des interfaces de programmation (API) web : entre science combinatoire et industrie du texte.* Theses, Paris IV Sorbonne, 2017, URL : https://tel.archives-ouvertes.fr/tel-01665406 (visité le 10/08/2022).

Interface de programmation, Wikipedia. L'encyclopédie libre, 2022, URL : https://fr.wikipedia.org/wiki/Interface_de_programmation (visité le 27/07/2022).

JANES (Andrea), REMENCIUS (Tadas), SILLITTI (Alberto) et SUCCI (Giancarlo), « Towards Understanding of Structural Attributes of Web APIs Using Metrics Based on API Call Responses », dans *Open Source Software : Mobile Open Source Technologies*, dir. Luis Corral, *et al.*, Series Title : IFIP Advances in Information and Communication Technology, Berlin, Heidelberg, 2014, t. 427, p. 83-92, DOI : 10.1007/978-3-642-55128-4_11.

MURDOCK (Jaimie) et ALLEN (Colin), « InPhO for All : Why APIs Matter », *Journal of the Chicago Colloquium on Digital Humanities and Computer Scienc*, 1–3 (2011), URL : http://www.jamram.net/docs/jdhcs11-paper.pdf (visité le 12/07/2022).

PRIME-CLAVERIE (Camille) et MAHÉ (Annaïg), « Le défi de l'interopérabilité entre plates-formes pour la construction de savoirs augmentés en sciences humaines et sociales », dans *Ecrilecture augmentée dans les communautés scientifiques*, 2017, URL : https://archivesic.ccsd.cnrs.fr/sic_01511618 (visité le 29/08/2022).

SMITH (David Neel) et BLACKWELL (Christopher W.), « Four URLs, limitless apps : Separation of concerns in the Homer Multitext architecture », dans *Donum natalicium digitaliter confectum Gre- gorio Nagy septuagenario a discipulis collegis familiaribus oblatum : A Virtual Birthday Gift Presented to Gregory Nagy on Turning Seventy by His Students, Colleagues, and Friends*, 2012, URL : https://chs.harvard.edu/d-n-smith-c-w-blackwell-four-urls-limitless-apps-separation-of-concerns-in-the-homer-multitext-architecture/ (visité le 22/08/2022).

WANG (Sean), BELOUIN (Pascal) et CHEN (Shih-Pei), « Research Infrastructure for the Study of Eurasia (RISE) : Towards a Flexible and Distributed Digital Infrastructure for Resource Access via Standardized Apis and Metadata », dans *DADH 2018. 9th International Conference of Digital Archives and Digital Humanities*, Taipei, 2018, p. 21-37, URL : https://pure.mpg.de/pubman/faces/ViewItemOverviewPage.jsp?itemId=item_3033461 (visité le 01/07/2022).

Marché de l'art, économétrie et statistiques

F-score, Wikipedia. L'encyclopédie libre, 2022, URL : https://en.wikipedia.org/wiki/F-score (visité le 12/06/2022).

MAUPEOU (Félicie Faizand de) et SAINT-RAYMOND (Léa), « Les "marchands de tableaux" dans le Bottin du commerce : une approche globale du marché de l'art à Paris entre 1815 et 1955 », *Artl@s Bulletin*, 2 (2013), URL : https://hal.archives-ouvertes.fr/hal-02986371 (visité le 20/06/1997).

PIKETTY (Thomas), *Les hauts revenus en France au XXe siècle : inégalités et redistributions, 1901-1998*, Paris, 2001.

Précision et rappel, Wikipedia. L'encyclopédie libre, 2022, URL : https://fr.wikipedia.org/wiki/Pr%C3%A9cision_et_rappel (visité le 13/06/2022).

Régression linéaire, Wikipedia. L'encyclopédie libre, 2022, URL : https://fr.wikipedia.org/wiki/R%C3%A9gression_lin%C3%A9aire (visité le 10/07/2022).

SAINT-RAYMOND (Léa), « Revisiting Harrison and Cynthia White's Academic vs. Dealer-Critic System », *Arts*, 8–3 (2019), DOI : https://doi.org/10.3390/arts8030096.

Glossaire

Open source L'*open source*, ou « code ouvert » définit à la fois un statut légal et un mouvement social. L'*open source* correspond à un logicel ou programme dont le code est librement consultable en ligne et réutilisable selon différentes licences plus ou moins permissives, comme la *GNU GPL v3.0* ou la *MIT License*. Ce statut légal a beaucoup gagné en popularité, en partie pour des raisons pragmatiques (un code ouvert est maintenu gratuitement, ce qui diminue les coûts d'entreprises) et symboliques (les GAFAM se sont beaucoup associées à ce courant en raison de l'image positive qu'il donne). Le statut est également associé à plusieurs mouvements en faveur du partage et de la réutilisation des connaissances, tel que la science ouverte. Certaines franges de l'*open source* sont plus radicales et ont une conception ouvertement politique de l'accessibilité au code, accessibilité qui permettrait aux utilisateur.ice.s de regagner en autonomie, de développer de nouveaux modèles communautaires et de reprendre le contrôle sur leurs comportements en ligne et leur consommation médiatique. Parmi les représentant.e.s les plus proéminent.e.s de ce courant se trouve Richard Stallman, créateur de la *GNU Foundation* et du système d'exploitation GNU, à la base de la majorité des systèmes d'exploitation utilisant Linux.. 68, 76

API Une interface de programmation d'application (*Application Programming Interface* en anglais) est un protocole qui pemret à un programme de communiquer avec un autre programme. Ce protocole documenté, correspond à un ensemble d'opérations permettant à un programme « consommateur » d'utiliser des fonctionnalités d'un programme « fournisseur », comme par exemple de récupérer ou d'envoyer des données au fournisseur.. 20

Base de données en graphe Une base de données en graphe sert à stocker des données reliées entre elles au sein d'un graphe ou d'un réseau. Les données correspondent aux nœuds du graphe et les relations sont représentées sous forme d'arrêtes. Souvent, données et relations disposent d'identifiants uniques définis grâce à des ontologies. L'interaction avec de telles bases de données est permise grâce à des languages de requêtes, comme `SPARQL`. La représentation en graphe a pour but de mettre l'accent sur l'interrelation entre les données et de permettre des liens complexes entre elles. [10].

10. *Graph database*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://en.wikipedia.org/`

83, 84, 89

CTS *Canonical Text Services* (CTS) est une API et une spécification pour identifier et diffuser des citations au sein de textes encodés. L'API spécifie la manière d'identifier un texte, un témoin, un passage ou encore une série de textes à l'aide d'un URN (un identifiant unique analogue aux URI). Le CTS est une API, mais ne spécifie pas comment une API CTS doit être implémentée : ce qui importe, c'est que l'API CTS permette d'accéder à des extraits de textes grâce à leurs citations [11].. 109

Dictionnaire Un dictionnaire est un format de données structuré en python qui associe à une donnée – dite clé – une ou plusieurs données nommées valeurs.. 25, 37, 40, 44, 45, 48, 50, 54, 63, 70–72, 121, 137

DTS Comme le CTS, le *Distributed Text Services* (DTS) est une API visant à identifier et permettre l'identification et la diffusion de texte dans une architecture client-serveur. Cette API, développée à partir de 2018 sur un modèle communautaire, demande une réponse en XML-TEI et des métadonnées encodées en JSON-LD ; ces spécifications techniques la distinguent de CTS, qui ne repose pas sur un format particulier. Comme le CTS, DTS ne définit pas comment implémenter l'API [12].. 109

Expression régulière Une expression régulière, ou expression rationnelle est une chaîne de caractère, écrite selon une syntaxe précise, qui permet de détecter des motifs dans du texte [13]. Les expressions régulières s'appuient sur la classification des caractères en classes (minuscules, majuscules, chiffres, espaces), sur des structures alternatives (un caractère ou un autre) et exclusives (un caractère n'ayant pas certaines propriétés) pour repérer des motifs. Par exemple, « 2022 » correspond au motif : `\d4`, soit « quatre chiffres à la suite ». Une adresse mail est également un motif : `[^(@|\s)]+@[^(@|\s)]+`, soit « plusieurs caractères qui ne sont ni un espace une arobase, suivi d'une arobase, suivi de plusieurs caractères qui n'est ni un espace ni une arobase ».. 24, 25, 35, 39, 50, 51, 54, 59, 60, 128, 145, 153

FAIR Les principes *Findable Accessible Interoperable Reusable* (FAIR) correspondent à un ensemble de préconisations simples pour le partage de données dans un contexte de sciences ouvertes [14].. 109

`wiki/Graph_database` (visité le 12/06/2022).

11. D. N. Smith et C. W. Blackwell, « Four URLs, limitless apps : Separation of concerns in the Homer Multitext architecture »…

12. B. Almas, T. Clérice, H. Cayless, *et al.*, *Distributed Text Services (DTS) : a Community-built API to Publish and Consume Text Collections as Linked Data*…

13. *Expression régulière*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://fr.wikipedia.org/wiki/Expression_r%C3%A9guli%C3%A8re` (visité le 10/07/2022).

14. M. Boeckhout, G. A. Zielhuis et A. L. Bredenoord, « The FAIR guiding principles for data stewardship… ».

Fichier de log Un fichier de log est un fichier qui est créé pendant l'exécution d'un algorithme pour sauvegarder des informations qui le concernent. Il s'agit en général d'informations annexes sur l'exécution du script : messages d'erreur, données produites en cours d'utilisation.... 71, 72, 77–79, 89, 90, 185

HTTP Le *Hypertext Transfer Protocol* (HTTP, ou « protocole de transfert hypertexte »), est un protocole qui permet la communication entre différentes machines dans une architecture client-serveur, comme celle du Web. C'est donc le protocole qui est à la base d'Internet, puisqu'il établit la manière dont des machines peuvent interagir au sein d'un réseau. Il définit un ensemble de méthodes pour interagir avec le serveur – demander (`GET`) des données, en envoyer (`POST`)... Il définit également la manière dont le serveur répond au client et un ensemble d'erreurs [15], chacune disposant d'un code pour l'identifier (l'erreur 404 indiquant que la ressource demandée n'existe pas, ou l'erreur 418 signifiant « Je suis une théière », un canular qui pouve l'humour des concepteur.ice.s du Web).. 67

Named Entity Linking (NEL) Le *Named Entity Linking*, traduit en français par « résolution d'entités nommées » ou « liage d'entités nommées » consiste à lier des entités – c'est-à-dire des objets uniques, comme des lieux, des personnes ou des organisations – à des sources de connaissance externes, comme des bases de connaissance en ligne.. 22

OAI-PMH L'OAI-PMH (*Open Archives Initiative Protocol for Metadata Harversting*) est un protocole notamment utilisé par les bibliothèques et les archives ; il permet de partager des métadonnées sur une collection de documents. Les données sont partagées en `XML` et s'appuient fortement sur l'usage du *Dublin Core*, un standard simple pour la structuration de métadonnées. Contrairement au CTS et au DTS, l'API OAI-PMH ne permet pas de partager un document, mais seulement ses métadonnées ; comme ces deux autres standards cependant, le standard OAI-PMH ne définit pas une implémentation [16].. 109

Ontologie Une ontologie correspond à la définition d'un ensemble de catégories, de propriétés et de relations qui unissent des données et des concepts ; cet ensemble est complété par une modélisation (souvent sous forme graphique), qui indique la relation entre les différents termes de l'ontologie [17]. Ceux-ci sont souvent liés de façon hiérarchique (plusieurs termes spécifiques pouvant être dérivés d'un terme générique). Dans le Web sémantique, chaque terme a lui-même un identifiant unique,

15. R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures…*, p. 116-127.

16. C. Prime-Claverie et A. Mahé, « Le défi de l'interopérabilité entre plates-formes pour la construction de savoirs augmentés en sciences humaines et sociales »…

17. *Ontology (information science)*…

ce qui garantit une implémentation uniforme pour l'ontologie et réduit le risque de présence de doublons.. 20

Python Python est un language de programmation impérative (c'est-à-dire, basé sur la production et la transformation de données par une suite d'instructions) créé en 1991. Il est particulièrement utilisé dans le domaine du traitement de données et des humanités numériques. C'est le langage le plus utilisé dans le projet *MSS / Katabase*. 120

REST L'architecture REST, définie par Roy Fielding (2000) [18] un standard pour le design et le fonctionnement interne d'API. Contrairement à d'autres standards (DTS, CTS, OAI-PMH), elle ne définit pas la manière de cibler des données grâce à des identifiants ; cette architecture définit plutôt comment l'interaction client-serveur doit être implémentée, à l'aide d'une série de principes de design. C'est le standard le plus répandu pour la conception d'API, bien qu'il soit souvent mal implémenté, ce qui a ammené au développement d'API *REST-like*.. 68

Score BLEU Un score BLEU (bilingual evaluation understudy) est un score servant à évaluer la qualité de la traduction d'un texte par une machine. Il est obtenu en comparant le texte traduit par une machine avec un plusieurs textes de référence (c'est-à-dire, des traductions faites par des humains du même texte).. 154

Score F1 Le score F1, ou *F-score*, est la moyenne harmonique de la précision (vrais positifs par rapport au total de résultats obtenus) et du rappel (nombre de résultats positifs par rapport au total de résultats positifs). [19]. Le score F1 a l'avantage de prendre en compte les vrais et les faux positifs. Ce score, dont la valeur est contenue entre 0 et 1, permet de mesurer l'exactitude d'un algorithme d'apprentissage machine, ou d'un moteur de recherche. Soit F le score F1, P la précision et R le rappel, sa formule est la suivante :

$$P = \frac{\text{items pertinents obtenus}}{\text{total des items obtenus}}$$

$$R = \frac{\text{items pertinents obtenus}}{\text{ensemble des items pertinents}}$$

$$F = 2 \times \frac{P \times R}{P + R}$$

. 25, 79, 80, 185

18. R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*...

19. *Précision et rappel*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://fr.wikipedia.org/wiki/Pr%C3%A9cision_et_rappel` (visité le 13/06/2022) ; *F-score*, Wikipedia. L'encyclopédie libre, 2022, URL : `https://en.wikipedia.org/wiki/F-score` (visité le 12/06/2022).

SPARQL SPARQL est un language de requête qui permet d'interagir avec une base de données au format RDF. « SPARQL exprime des requêtes sur des sources de données diverses [...]. SPARQL rend possible la requête de données en graphes [...], avec des données conjointes et disjointes. SPARQL permet également l'agrégation de données, les sous-requêtes, la négation, la création de données à l'aide d'expression, le test de données et la contrainte des requêtes. Les résultats de requêtes sparql peuvent être des jeux de résultats ou des graphes RDF. » [20]. 13, 20, 21, 23–25, 27, 65, 83–96, 98, 100, 124, 132, 143, 213, 217, 224, 263, 271, 277

Table de conversion Une table de conversion est, tout simplement, un dictionnaire qui contient en clés un certain nombre d'informations telles qu'elles figurent dans le texte et, en valeurs, une version normalisée de cette information. Cela permet de détectier des motifs à extraire autant de normaliser les informations.. 25, 35, 39

URI Une URI (*Uniform Resource Identifier*) est une chaîne de caractères qui identifie une ressoource sur un réseau. Cette URI est permanente, et permettra d'identifier une ressource même si celle-ci est supprimée. L'URL est un type particulier d'URI qui, en plus de d"crire une resource, la représente aussi : un URL de page *Wikipedia* permet d'identifier une page de façon unique, comme toute URI (aucune autre page sur internet n'ayant le même URL) ; mais en plus, elle pêermet d'accéder à la page elle-même.. 21

URL Une URL (*Uniform Resource Locator*) est un identifiant unique sous la forme d'une chaîne de caractères pointant vers une ressource sur le Web. De la même manière qu'une cote de bibliothèque pointe vers un livre précis, un URL pointe vers une page ou une donnée sur le Web. L'URL permet d'agir sur une ressource (en envoyant des données...) et peut être associée à des données ainsi qu'à des reésentations (une page Web, par exemple).. 69

20. « SPARQL can be used to express queries across diverse data sources [...]. SPARQL contains capabilities for querying [...] graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs. »Steve Harris, Andy Seaborne et Eric Prud'hommeaux, *SPARQL 1.1 Query Language. W3C Recommendation 21 March 2013*, W3C, 2013, URL : `http://www.w3.org/TR/2013/REC-sparql11-query-20130321/` (visité le 10/06/2022) (traduction de l'auteur).

Acronymes

API Application Programming Interface. 20, 22, 23, 25, 30, 32, 33, 36, 37, 40, 45, 53, 56, 57, 59, 62, 65–72, 75–77, 79, 101, 107–109, 111–126, 128–131, 134, 136, 137, 140–146, 151, 211, 271, 276, 277, *Glossaire :* API

CTS *Canonical Text Services*. 109, 115–119, *Glossaire :* CTS

DTS *Distributed Text Services*. 109, 115, 118, 119, *Glossaire :* DTS

FAIR *Findable Accessible Interoperable Reusable*. 109–111, 114, 121, 130, 132, 135, 140, 143, 144, 151, *Glossaire :* FAIR

HTTP *Hypertext Transfer Protocol*. 67, 69, 89, 107, 112, 113, 123, 124, 131, 133, 142, 146–149, *Glossaire :* HTTP

NEL Named Entity Linking (NEL). 22, 23, 79, 80, 101, 187, 273, *Glossaire :* Named Entity Linking (NEL)

OAI-PMH *Open Archives Initiative Protocol for Metadata Harversting*. 109, 115, 116, 119, *Glossaire :* OAI-PMH

REST *Representational state transfer*. 68, 109, 112, 114, 116, 121, 122, 124, 128–132, 136, 141, 143, 144, 151, *Glossaire :* REST

URI *Uniform Resource Identifier*. 21, 92, 93, 112, 117, 118, *Glossaire :* URI

URL *Uniform Resource Locator*. 69–72, 98, 100, 107–109, 112, 117, 119, *Glossaire :* URL

Table des figures

Liste des tableaux

Table des matières